WebSphere software

IBM

IBM WebSphere Transformation Extender 8.2

- Includes key concepts and functionality within WebSphere Transformation Extender
- Covers all related editions on distributed and z/OS platforms
- Provides integration scenarios

John Bergland
G. Michael Connolly
Gerrit De Troyer
Piotr Dziedzic
Alain Manteau
Philippe Mongrolle
Patricia Pettersson
Brian E. Wilson

Redbooks

ibm.com/redbooks

International Technical Support Organization

IBM WebSphere Transformation Extender 8.2

February 2009

SG24-7693-00

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

First Edition (February 2009)

This edition applies to IBM WebSphere Transformation Extender 8.2.

Contents

Notices xi
Trademarks xii

Preface xiii
The team that wrote this book xiii
Become a published author xvi
Comments welcome xvi

Part 1. Introduction to WebSphere Transformation Extender 1

Chapter 1. Overview of WebSphere Transformation Extender 3
1.1 WebSphere Transformation Extender: Enabling universal transformation . 4
1.1.1 Business value of WebSphere Transformation Extender 5
1.2 WebSphere Transformation Extender Design Studio 7
1.2.1 Type Tree Editor 8
1.2.2 Map Editor 9
1.2.3 Database Interface Designer 9
1.2.4 Integration Flow Designer 10
1.2.5 Using WebSphere Transformation Extender Design Studio with DataPower SOA appliances 10
1.3 WebSphere Transformation Extender Online Library 11
1.4 Overview of WebSphere Transformation Extender Editions 11
1.4.1 WebSphere Transformation Extender with Command Server 11
1.4.2 WebSphere Transformation Extender with Launcher 12
1.4.3 WebSphere Transformation Extender for Integration Servers 12
1.4.4 WebSphere Transformation Extender for Application Programming 14
1.4.5 WebSphere Transformation Extender SDK 14
1.5 WebSphere Transformation Extender on System z 14
1.5.1 WebSphere Transformation Extender with Command Server on System z 16
1.5.2 WebSphere Transformation Extender with Launcher on System z 16
1.5.3 WebSphere Transformation Extender for Integration Servers on System z 17
1.5.4 WebSphere Transformation Extender for Application Programming on System z 17
1.5.5 WebSphere Transformation Extender on Linux on System z 17
1.6 WebSphere Transformation Extender with other products 18
1.6.1 Lotus Forms 18

1.6.2 IBM InfoSphere Information Server 19

Chapter 2. Overview of the industry solutions for WebSphere Transformation Extender 21
2.1 Industry packs 22
2.1.1 Healthcare 23
2.1.2 Financial services 24
2.1.3 Insurance 25
2.1.4 EDI 26
2.2 Enterprise packs 27
2.2.1 WebSphere Transformation Extender for PeopleSoft 27
2.2.2 WebSphere Transformation Extender for SAP R/3 28
2.2.3 WebSphere Transformation Extender for SAP XI 29
2.2.4 WebSphere Transformation Extender for Siebel 30
2.3 Adapters 31
2.3.1 JCA adapters 32
2.3.2 Adapters on System z 32
2.4 WebSphere Transformation Extender Extended Functions 33
2.4.1 IBM WebSphere Transformation Extender Secure Adapter Collection 33
2.4.2 WebSphere Transformation Extender SNMP Collection 33
2.5 Trading Manager 34

Part 2. WebSphere Transformation Extender concepts and functionality 37

Chapter 3. Basic concepts of WebSphere Transformation Extender 39
3.1 Design process 40
3.2 Type trees 42
3.2.1 Exploring type trees 43
3.2.2 Importers 69
3.2.3 Type Tree Maker 71
3.3 Maps 72
3.3.1 Map files 72
3.3.2 Cards 73
3.3.3 Rules 78
3.3.4 Functional maps 82
3.3.5 Validation maps 84
3.4 Systems 86
3.4.1 System components 86
3.4.2 Preparing systems to run 89
3.4.3 Launcher 90

Chapter 4. Methods of execution 93
4.1 WebSphere Transformation Extender runtime editions 94

4.2 WebSphere Transformation Extender with Command Server 96
4.2.1 Installing WebSphere Transformation Extender with Command Server on distributed systems 97
4.2.2 Using WebSphere Transformation Extender with Command Server on distributed systems 98
4.2.3 Running a map in WebSphere Transformation Extender with Command Server on z/OS systems 101
4.3 WebSphere Transformation Extender with Launcher 102
4.3.1 Installing WebSphere Transformation Extender with Launcher . . . 104
4.3.2 Using WebSphere Transformation Extender with Launcher 106
4.4 WebSphere Transformation Extender for Integration Servers with WebSphere Message Broker 128
4.4.1 Installing WebSphere Transformation Extender for Integration Servers with WebSphere Message Broker 129
4.4.2 Using WebSphere Transformation Extender for Integration Servers with WebSphere Message Broker 135
4.5 WebSphere Transformation Extender for Integration Servers with WebSphere ESB and WebSphere Process Server 152
4.5.1 Installing WebSphere Transformation Extender for Integration Servers with WebSphere ESB and WebSphere Process Server. 153
4.5.2 Using WebSphere Transformation Extender for Integration Servers with WebSphere ESB and WebSphere Process Server 155
4.6 WebSphere Transformation Extender for Application Programming edition APIs 165
4.6.1 Installing WebSphere Transformation Extender for Application Programming edition APIs 166
4.6.2 Using WebSphere Transformation Extender for Application Programming edition APIs 167

Part 3. Getting started: Implementation using WebSphere Transformation Extender . . 169

Chapter 5. Developing in Design Studio 171
5.1 Installing WebSphere Transformation Extender Design Studio 172
5.2 Design Studio language 180
5.3 Using Design Studio 182
5.3.1 Projects 200
5.3.2 Importing and building type trees 204
5.3.3 Analyzing type trees 237
5.3.4 Mapping 240
5.3.5 Map settings 270
5.3.6 Compiling maps 276
5.3.7 Integration Flow Designer 280
5.3.8 Database Interface Designer 296

5.3.9 Unit testing 318
5.3.10 Tracing and debugging 320
5.3.11 Adapters: The GET and PUT functions 330
5.3.12 Resource Registry 332
5.4 Command line utilities 349

Chapter 6. WebSphere Transformation Extender on System z 353
6.1 Installation on z/OS 354
6.1.1 Prerequisites for running WebSphere Transformation Extender on z/OS: XML Toolkit v1.9 355
6.2 Transformation and execution on z/OS 356
6.2.1 z/OS administration 356
6.2.2 z/OS adapters 358
6.3 Running a map in batch mode 358
6.3.1 Building a map in Design Studio for the z/OS platform 360
6.3.2 Transferring a compiled map to a z/OS system 362
6.3.3 Transferring data files to a z/OS system 363
6.3.4 Modifying or creating a JCL file to execute as a batch job 367
6.3.5 Submitting a job 380
6.3.6 Examining the results 380
6.4 Focus on WebSphere Transformation Extender for z/OS 383
6.4.1 Input data validation: Error handling by using the restart flag and REJECT function 383
6.4.2 Reading and writing XML data 384
6.4.3 Packed decimal data 392
6.4.4 Code page conversion fallback 392
6.4.5 The convert function 394
6.4.6 Zero execution return code in JCL 395
6.4.7 The run map function 396
6.4.8 Dynamic DD references with a PDS (member) mechanism 397
6.4.9 CPACKAGE and CTEXT functions in the RUN, GET, and PUT functions 398
6.4.10 The z/OS DB2 adapter 399
6.4.11 The EXIT function 400
6.4.12 VSAM considerations 402
6.4.13 The Resource Registry 402
6.4.14 The DTXPAGE utility 405
6.4.15 The DTXPROF utility 408
6.5 WebSphere Transformation Extender for z/OS: Application Programming 413
6.5.1 The Platform APIs on z/OS 413
6.5.2 The SDK on UNIX System Services 420

6.6 WebSphere Transformation Extender for z/OS: CICS 423
6.7 WebSphere Transformation Extender for z/OS: IMS TM and IMS/DC . . 424
6.8 WebSphere Transformation Extender for z/OS: Command Server, Launcher, and SDK for UNIX System Services 425
6.8.1 Post installation in UNIX System Services 427
6.8.2 Running a map 431
6.9 WebSphere Transformation Extender on Linux on System z 432
6.10 WebSphere Transformation Extender for z/OS: Integration servers . . . 437
6.10.1 WebSphere Transformation Extender and WebSphere Message Broker on z/OS 437
6.10.2 WebSphere Transformation Extender and WebSphere Process Server or WebSphere ESB 440

Chapter 7. Troubleshooting 443
7.1 Approach to troubleshooting 444
7.2 Validation 444
7.2.1 Data validation 445
7.3 Additional troubleshooting techniques 460
7.3.1 Summary Trace setting 461
7.3.2 Keywords 462
7.3.3 Locating the last error and working backwards 462
7.3.4 Offset value 463
7.3.5 Comparing valid and invalid data from a trace 463
7.3.6 Map data audit log 464
7.3.7 Error audit log status codes 464
7.3.8 Map Debugger 465
7.4 Error handling 468
7.4.1 Restart attribute 468
7.4.2 REJECT function 471

Part 4. WebSphere Transformation Extender integration scenarios 473

Chapter 8. Integration scenarios: WebSphere Transformation Extender for Integration Servers 475
8.1 Business scenario introduction 476
8.1.1 The process for scenario 1: WebSphere ESB 477
8.1.2 The process for scenario 2: WebSphere Message Broker 478
8.1.3 The MT103 message 480
8.1.4 Web services descriptions 482
8.2 Available resources and the components that are built in each scenario 483
8.2.1 Available resources 483
8.2.2 The components that are built in each scenario 489
8.3 Building the WebSphere ESB scenario 490
8.3.1 Roadmap for building the WebSphere ESB scenario 491

8.3.2 Preparing the artifacts 491
8.3.3 Creating the WebSphere Transformation Extender maps 496
8.4 Creating the first WebSphere Transformation Extender map 504
8.4.1 Creating the type trees 504
8.4.2 Creating the validation map 511
8.4.3 Creating the SWIFT to XML transformation map 515
8.4.4 Creating the XML to SWIFT executable map 523
8.4.5 Building the WebSphere ESB scenario. 526
8.4.6 Deploying and testing the scenario on WebSphere ESB 578
8.5 Building the WebSphere Message Broker Message scenario 580
8.5.1 Roadmap for building the WebSphere Message Broker scenario . 581
8.6 Creating the second WebSphere Transformation Extender map 582
8.6.1 Building the WebSphere Message Broker scenario 591
8.6.2 Deploying and testing the scenario on WebSphere Message Broker 611

Chapter 9. Integration scenario: WebSphere Transformation Extender on System z 623
9.1 Scenario introduction. 624
9.2 Building the scenario 627
9.2.1 Creating the project. 629
9.2.2 Creating the type trees 629
9.2.3 Creating the error handling type tree 645
9.2.4 Creating the executable map 648
9.2.5 Creating the functional maps 660
9.2.6 Optional: Creating the audit log map. 662
9.2.7 Building and running the map for unit testing 671
9.2.8 Building and running the error handling map 674
9.2.9 Building the maps for z/OS 675
9.2.10 Running the maps on z/OS 676

Appendix A. Running on z/OS 679
Submitting a JCL 680
Exploring the project on z/OS. 681
Submitting the job and reviewing the job results 698

Appendix B. Running the test cases for the WebSphere Message Broker scenario 713
Overview of the test cases 714
Test case 1. 715
Test case 2. 718
Test case 3. 720
Test case 4. 722
Test case 5. 724

Test case 6 726
Test case 7 728
Test case 8 730
Test case 9 732

Appendix C. Using type trees from the SWIFTNet FIN industry pack . . 735
Creating a new type tree with the SWIFT generic ISO 7775 type tree from the SWIFTNet FIN Industry Pack 736
Using the SWIFT generic ISO 7775 type tree with a template type tree 739

Appendix D. z/OS SDTXSAMP contents 741

Appendix E. Additional material 749
Locating the Web material 749
Integration scenario 1: WebSphere Transformation Extender for Integration Servers with WebSphere ESB 750
Integration scenario 2: WebSphere Transformation Extender for Integration Servers with WebSphere Message Broker 751
Integration scenario 3: WebSphere Transformation Extender on System z . 752
Importing the Project Interchange file 753

Related publications 755
Other publications 755
Online resources 755
How to get Redbooks 756
Help from IBM 756

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.

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:

AIX®	Domino®	OS/390®
CICS®	IBM®	Redbooks®
DataPower device®	Informix®	Redbooks (logo) ®
DataPower®	InfoSphere™	System z®
DataStage®	Lotus Notes®	Tivoli®
DB2 Universal Database™	Lotus®	WebSphere®
DB2®	MQSeries®	z/OS®
developerWorks®	Notes®	z/VM®

The following terms are trademarks of other companies:

Acrobat, Adobe, and Portable Document Format (PDF) are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Snapshot, and the NetApp logo are trademarks or registered trademarks of NetApp, Inc. in the U.S. and other countries.

SUSE, the Novell logo, and the N logo are registered trademarks of Novell, Inc. in the United States and other countries.

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

BAPI, SAP R/3, SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

EJB, Enterprise JavaBeans, eXchange, J2EE, Java, JavaBeans, JNI, JRE, JVM, Solaris, Sun, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

BizTalk, Expression, Microsoft, MS, PowerPoint, SQL Server, Visual SourceSafe, Windows Vista, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Itanium, 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.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Preface

This IBM® Redbooks® publication provides an introduction to IBM WebSphere® Transformation Extender V8.2. WebSphere Transformation Extender is a powerful, transaction-oriented universal data transformation and validation solution. It offers multiple execution options to support right-time, right-style transformation, whether it is batch, real-time, or embedded.

In this Redbooks publication, we present an overview of the WebSphere Transformation Extender products and components and discuss key features for businesses. We describe both the development and runtime or management aspects of the product and review troubleshooting capabilities. This book is written for developers, architects, and IT decision makers who desire a thorough understanding of the various editions and features of WebSphere Transformation Extender.

In this book, we discuss the WebSphere Transformation Extender role in a service-oriented architecture (SOA) working with other components of the IBM Software portfolio, both on distributed platforms and on the IBM System z® platform. We include hands-on scenarios that show the end-to-end process to create, deploy, and execute WebSphere Transformation Extender. The scenarios show how WebSphere Transformation Extender works with IBM WebSphere Enterprise Service Bus (ESB) and WebSphere Message Broker. They also show how WebSphere Transformation Extender works on IBM z/OS® environement. This information can help you to understand the product framework as you get started.

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 (ITSO), Poughkeepsie Center.

John Bergland is a project leader at the ITSO, Cambridge Center. He manages projects that produce Redbooks publications that focus on IBM WebSphere and Lotus® software technology. Before joining the ITSO in 2003, John worked as an IT Specialist with IBM Software Services for Lotus, specializing in Lotus Notes® and Domino® messaging and collaborative solutions. He holds a Master of Business Administration (MBA) degree and a Master of Science (MS) degree in information technology from Boston University.

G. Michael Connolly is an IT consultant at the ITSO, Poughkeepsie Center. He has more than 30 years of IBM software development experience in both distributed systems and the mainframe System z platform. He holds a Bachelor of Arts (BA) in Humanities from Villanova University. His areas of expertise include TCP/IP communications, UNIX® System Services, and WebSphere Application Server for z/OS.

Gerrit De Troyer is a Certified IT Specialist in Belgium and Luxembourg and has been with IBM since 1997. He has worked in several roles and is now a technical presales specialist for WebSphere. He has extensive experience in a wide range of technologies and products including Java™ and J2EE™, WebSphere MQ, WebSphere Message Broker, WebSphere ESB, and WebSphere Transformation Extender. Gerrit holds an engineering degree in electronics.

Piotr Dziedzic is a Software Services Consultant in Poland. He has over 10 years of experience in the IT field, working in several roles and specializations. His areas of expertise include WebSphere Transformation Extender, WebSphere Partner Gateway, WebSphere Process Server, and WebSphere ESB. He holds an MS degree in computer science from Jagiellonian University in Krakow.

Alain Manteau is a Certified IT Specialist in France. He has more than 35 years of experience in the IT field. His areas of expertise include Decision Support Systems. Alain joined IBM in 2000. He has worked in several roles and is now a technical presales specialist for the WebSphere in Software Group z Series team. He has extensive experience in a wide range of technologies and products including WebSphere Data Interchange, WebSphere Message Broker, and WebSphere Transformation Extender.

Philippe Mongrolle is the CIO of Satisco SA in Luxembourg (satisco http://www.satisco.lu), which is an advanced business partner of IBM specialized on WebSphere Transformation Extender. Philippe is a certified trainer and has been developing and managing WebSphere Transformation Extender projects since 2001 in numerous client engagements worldwide. He holds an MBA degree from Dauphine University, France.

Patricia Pettersson is an IT Specialist in Sweden. She has more than 20 years of experience in the IT field, having worked in several roles and specializations. Her areas of expertise include WebSphere Transformation Extender and WebSphere Partner Gateway. Patricia holds an MBA degree in electrical engineering from UNICAMP, Brazil.

Brian E. Wilson is a Senior Certified Executive IT Specialist in the United States in the Greater New York region. He has 26 years of experience in the IT field, covering a large variety of specializations and roles. Brian joined IBM in 2000 as a technical specialist for WebSphere. His areas of expertise include WebSphere

MQ, WebSphere Message Broker, WebSphere Partner Gateway, WebSphere ESB, and WebSphere Transformation Extender.

The team from left to right: Brian Wilson, G. Michael Connolly, Gerrit De Troyer, Patricia Pettersson, Piotr Dziedzic, Philippe Mongrolle, Alain Manteau, and John Bergland

Thanks to the following people for their contributions to this project:

- Laurent Barthelemy, IBM Software Group, Application and Integration Middleware Software, IBM France
- Paul A. Brett, IBM WebSphere Transformation Extender Customer Support, IBM U.K.
- Richard M. Conway, ITSO Residency Enablement for z/OS and IBM System z and p, IBM U.S.
- Kimberly Drouot, WW Transformation and Industry Standards Sales Leader, IBM France
- Michael J. Hudson, Software Architect, WebSphere Transformation Extender, IBM Software Group, IBM U.S.
- Emmanuel Jambou, IBM WebSphere Transformation Extender Customer Support, IBM France
- Dan Pitre, Product Manager - WebSphere Transformation Extender, IBM Software Group, IBM Canada

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

- Mail your comments to:

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

Part 1

Introduction to WebSphere Transformation Extender

In this part, we introduce you to WebSphere Transformation Extender and all of its components. This part includes the following chapters:

- Chapter 1, “Overview of WebSphere Transformation Extender” on page 3
- Chapter 2, “Overview of the industry solutions for WebSphere Transformation Extender” on page 21

1

Overview of WebSphere Transformation Extender

In this chapter, we introduce WebSphere Transformation Extender. We describe the development toolset and explain each of the different editions of WebSphere Transformation Extender. We include information about the use of WebSphere Transformation Extender on the IBM System z platform.

1.1 WebSphere Transformation Extender: Enabling universal transformation

The WebSphere Transformation Extender product is a powerful, transaction-oriented universal data transformation and validation solution that delivers flexibility for your IT systems, resulting in business agility. It automates the transformation of high-volume, complex transactions without the need for hand-coding, which provides enterprises with a quick return on investment.

WebSphere Transformation Extender performs transformation and routing of data, in any format including XML, non-XML, and mixed formats, from source systems to target systems in batch and real-time environments. The sources can include files, relational databases, message-oriented middleware (MOM), packaged applications, or other external sources. After retrieving the data from its sources, WebSphere Transformation Extender transforms the data and routes it to any number of targets where it is needed, providing the appropriate content and format for each target system.

WebSphere Transformation Extender is integrated with other components of the IBM WebSphere integration software suite for complete operational and transactional data integration across the enterprise. It provides the following benefits:

- Highly automated transformation and routing of complex data across many points of integration in real time to support high message volumes
- Enterprise-wide interoperability supported by a services-oriented architecture (SOA) for seamless connectivity and interoperability across back-office systems
- Connectivity to a wide range of mainframe, existing, and enterprise applications, databases, messaging systems, and external information sources
- Support for high-performance, event-driven, transactional environments to ensure completion and validation of transactions in real time
- A comprehensive library of more than 120 pre-built functions to reduce development time and simplify specification of rules for validation, transformation, and routing
- Seamless integration across the development, data, and production layers of the enterprise, leveraging existing IT infrastructures
- Ready-to-use solutions for industry standards and regulatory compliance for operational and transactional data integration

- Multiple execution options to support right-time, right-style transformation, whether it is batch, real-time, or embedded

WebSphere Transformation Extender exposes transformations as services inside SOAs. The WebSphere Transformation Extender family delivers editions for both batch- and event-driven systems, extending an enterprise service bus (ESB) or business process management (BPM) solution. The editions can be embedded with customer applications written in a variety of languages including C, COBOL, Java, and .Net. WebSphere Transformation Extender supports many platforms, including the System z platform, where customers can deploy WebSphere Transformation Extender in z/OS batch, CICS®, IMS, UNIX System Services, and Linux® on System z environments.

1.1.1 Business value of WebSphere Transformation Extender

Without a dedicated solution for transformation, you need significant programming skills to transform complex formats, such as a Health Insurance Portability and Accountability Act (HIPAA), electronic data interchange (EDI), or an SAP® IDOC document to other formats. In this scenario, you need to write specialized, format-specific programs to perform the preprocessing and postprocessing tasks of checking content quality. The tasks that are associated with manually transforming formats must all be administered and maintained over time, incurring development and maintenance costs.

WebSphere Transformation Extender minimizes the impact that integration imposes on an application. WebSphere Transformation Extender maintains each transformation mapping, including content validation and format variations, in a portable form, regardless of sources, target applications, platforms, or adapter requirements, and without the user having to write code. Its powerful write-once, deploy-anywhere portable technology enables you to grow your infrastructure with your business.

New business services can be delivered quicker, with business-driven enhancements turned around in shorter cycles. Business is also better served when electronic documents that are exchanged between organizations can be validated before they are sent or received, eliminating the costly errors and corrective actions that are required to resend the documents. Validation checking is also important for businesses that must comply with regulatory and industry requirements for business-to-business transactions. In-process validation of data before it is mapped eliminates wasted processing time, which can be expensive.

What makes WebSphere Transformation Extender so different? WebSphere Transformation Extender has unique capabilities that make it extremely powerful:

- WebSphere Transformation Extender has a unique many-to-many model of transforming and processing data. With this model, WebSphere Transformation Extender can run all transforms, lookups, and data enrichments with only one pass at the data, making it one of the best performing transformation engines on the market.
- Data is validated to content rules and context usages as part of the transformation process. It is not necessary to write separate logic or have separate executions to provide extremely rich data validation.
- There is no “language” to WebSphere Transformation Extender. The transforms and data process are all maintained within the spreadsheet-type GUI. It is not necessary to write code to handle complex transforms.
- WebSphere Transformation Extender uses self-describing data models to handle data in its native format and has a unique mechanism for describing data in its native form.

Figure 1-1 summarizes the types of problems that WebSphere Transformation Extender excels at solving.

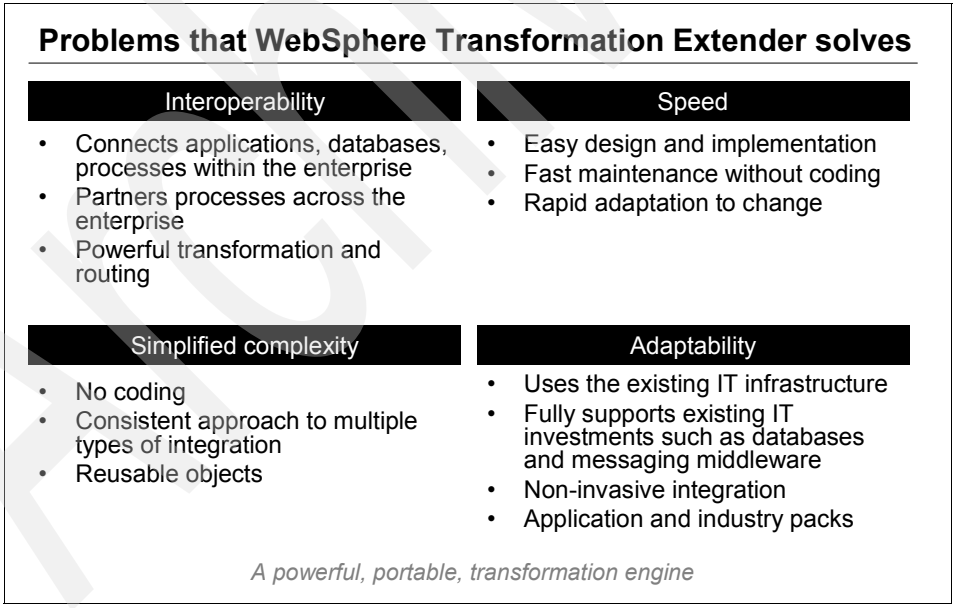

Figure 1-1 Problems solved by WebSphere Transformation Extender

1.2 WebSphere Transformation Extender Design Studio

The WebSphere Transformation Extender Design Studio components are the design-time applications that are used to model, develop, manage, and test data transformations, application integration maps, and systems in a development environment. Design Studio provides the means for developing event-driven application-to-application integration, business-to-business integration, and consumer-to-business application integration. Integration developers and data architects use Design Studio to visualize both simple and complex data types.

You can use Design Studio to complete the following tasks:

- Define your data
- Create maps for your data
- Manage the systems of maps that you create
- Develop and test maps and systems in a Microsoft® Windows® environment

Design work can be centralized or distributed. A single Design Studio can serve multiple servers, or multiple Design Studios can be used to create interfaces for a single production environment.

Design Studio includes several components:

Type Tree Editor The modeling component that is used to define data objects, including source and target data structures.

Map Editor The modeling component that is used to develop maps that define your data transformation logic.

Database Interface Designer
The modeling component that is used to import metadata about queries, tables, and stored procedures for data stored in relational databases. The Database Interface Designer identifies characteristics, such as update keys and database triggers, of those objects to meet mapping and execution requirements.

Integration Flow Designer
The modeling component that is used to define and manage data integration processes. Use the Integration Flow Designer to define interactions among maps and systems of maps, to validate the logical consistency of workflows, and to prepare systems to run.

The Type Tree Editor and Map Editor are Eclipse-based tools. The Eclipse workbench delivers a common, standards-based user experience across the WebSphere portfolio, and is supported by many independent software vendors.

XML schemas: The Design Studio has strong support for XML schemas. This support includes the use of an XML schema directly in a map, switchable on/off XML schema validation on a per-card basis. It also includes support for the XSD Union type and for the xsi:type attribute.

Transformations and validations can be fully developed, analyzed, and tested on a developer's workstation by using Design Studio, before generating a platform-specific compiled format for use with any WebSphere Transformation Extender edition engine. As a file-based solution, no additional database repository is required to manage the transformation artifacts. Any preferred Eclipse-supported source-code management solution can be used to manage the assets between development teams.

1.2.1 Type Tree Editor

The Type Tree Editor is the design component that is used to specify, define, and manage type definitions in the form of type trees. A *type tree* is a data dictionary that defines how types are classified. The type tree files that you create in the Type Tree Editor are data definition files. Each object identified in your data is defined as a *type* in your type tree. Type trees can also be created using the Type Tree Maker, the Importer Wizard, and the Database Interface Designer.

You can use the Type Tree Editor to define the following information:

- Properties for data structures
- Data validation rules
- Data as text or binary
- Different character sets

A type tree describes the syntax, structure, and semantics of your data. The *syntax* of data refers to its format. The format of data includes tags, delimiters, terminators, and other characters that separate or identify data breaks. The *structure* of data refers to its composition, including repeating substructures, nested groupings, sequences, and choices. The *semantics* of data refer to the meaning of the data, including rules for data values, relationships among parts of a large data object, and error detection and recovery.

The Type Tree Editor includes an importer tool for automatically generating type trees for data that is described in XML, COBOL, and other formats or application data structures.

1.2.2 Map Editor

The Map Editor is an application in Design Studio that is used to specify data transformation logic in the form of map rules. The Map Editor uses the definition of data objects that are created in the Type Tree Editor as inputs and outputs. The Map Editor provides the functionality to specify rules for the transformation and routing of data. Maps are analyzed, compiled, and tested by using the Map Editor.

This process is facilitated by convenient “from” and “to” windows, drag-and-drop techniques, and spreadsheet-type rules. Mapping rules are added by using logical statements. The Map Editor provides a rich set of more than 100 predefined functions for operations such as conditional testing, table lookups, mathematical functions, character string parsing, date and time handling, and data extraction.

The Map Editor is used to perform the following tasks:

- Create maps to specify the logic necessary for transforming the input data to the desired output data
- Identify the source and data objects of the input data
- Identify the target and data objects of the output data
- Specify and build the output data according to the map rules
- Provide information about data validation by generating trace files
- View the run results of the map execution

You build and run maps by using the Map Editor on a Windows platform. You can also use the Map Editor to build a map to be run on another platform. Each map can be built for a specific platform and then executed on that platform to perform the transformation of the data.

1.2.3 Database Interface Designer

The Database Interface Designer is used to import metadata about queries, tables, and stored procedures for data that is stored in relational databases. Use the Database Interface Designer to identify characteristics of those objects to meet mapping and execution requirements such as update keys and database triggers.

Use the Database Interface Designer to perform the following tasks:

- Specify the databases to use for a source or target
- Define query statements
- Automatically generate type trees for queries or tables

1.2.4 Integration Flow Designer

The WebSphere Transformation Extender Integration Flow Designer is the modeling component that is used to define and manage data integration processes. The Integration Flow Designer is used to define interactions among maps and systems of maps, to validate the logical consistency of workflows, and to prepare systems of maps to run.

Collections of maps are called *systems*. Systems can be used as a focal point for interfacing to other development components and for preparing maps for execution. The systems created and managed in the Integration Flow Designer can be defined with existing maps, or they can be designed with maps that have not yet been created (pseudo map components). Systems can also be designed with subsystems.

Use the Integration Flow Designer to perform the following tasks:

- Define interactions among maps and systems of maps
- Specify the integration workflow
- Display data flow relationships among system components
- Validate the logical consistency of workflows
- Prepare systems to run
- Verify component relationships
- Generate process control information
- Experiment at design time
- Coordinate multiple data sources and targets
- Coordinate events
- Model a variety of what-if scenarios
- Visualize the scenario results
- Create and manage collections of logically related maps
- Specify system deployment servers

1.2.5 Using WebSphere Transformation Extender Design Studio with DataPower SOA appliances

WebSphere Transformation Extender Design Studio can be used to design maps for processing on WebSphere DataPower® XI50 Appliances. WebSphere DataPower devices provide powerful security, XML acceleration, and ESB functionality in an easily maintainable environment. Design Studio provides a highly productive way of defining formats that are accepted by the DataPower appliance and, similarly, for defining the maps between those formats.

You must only modify one setting (map run time), and then you are ready to *design* for the WebSphere DataPower environment. By using the build map operation inside Design Studio, you can verify that all of the mapping rules are

supported by the WebSphere DataPower run time. You can also unit test a map on the DataPower device® from Design Studio.

All of the mapping concepts that you are familiar with for WebSphere Transformation Extender, such as type trees and maps, apply in WebSphere DataPower, making the transition as seamless as possible.

1.3 WebSphere Transformation Extender Online Library

WebSphere Transformation Extender provides an online library, in addition to the Design Studio Help system and the information center (available both on the IBM Web site and installed with Design Studio). The WebSphere Transformation Extender Online Library is a set of reference manuals (provided as Adobe® Acrobat® PDF files) that cover the many different components and major functions of WebSphere Transformation Extender.

The online library is installed separately. You should install the online library last, after you have installed all WebSphere Transformation Extender components. When you install the online library, you can choose to either install the PDF files on your system or access them from a CD-ROM.

1.4 Overview of WebSphere Transformation Extender Editions

WebSphere Transformation Extender is packaged in different editions to suit the deployment mode that best fits your requirement:

- As a batch-file processing engine called from a command script, batch or command file, or System z job control language (JCL)
- As an event-driven server, either stand-alone or as an ESB or BPM extender
- As an embedded engine inside any COBOL, C, or Java application

1.4.1 WebSphere Transformation Extender with Command Server

By using WebSphere Transformation Extender with Command Server, you can run transformations directly from a command line, shell script, timer, or JCL on System z. You can extend an existing application and infrastructure with minimal intrusion. WebSphere Transformation Extender with Command Server for System z supports MVS batch, CICS, and IMS environments, and includes the

components in the WebSphere Transformation Extender for Application Programming edition.

1.4.2 WebSphere Transformation Extender with Launcher

WebSphere Transformation Extender with Launcher provides a stand-alone event server to host the WebSphere Transformation Extender engine. It includes the command line and script processing capabilities that are in WebSphere Transformation Extender with Command Server.

The Launcher activates transformations when triggered by an event, which might be a file creation, a message arriving on a WebSphere MQ queue, a database trigger, a scheduler, and many other options including combinations of triggers. The Launcher offers an online run-time environment that synchronizes and controls complicated data transformations while maintaining and optimizing the execution environment for the transformations.

WebSphere Transformation Extender with Launcher can handle complex events and support time- and availability-based event synchronization on combinations of triggers on multiple input sources. For the System z platform, this edition runs under the UNIX System Services environment.

1.4.3 WebSphere Transformation Extender for Integration Servers

WebSphere Transformation Extender for Integration Servers provides extenders to other WebSphere products, providing installation and integration support. WebSphere Transformation Extender complements the native mediation and business-rule capabilities of these products and can process large documents and messages with more complex formats that are not based on XML. By using WebSphere Transformation Extender, you can solve critical business problems in real time with support for industry standards and regulatory compliance.

The following WebSphere Transformation Extender products are packaged as WebSphere Transformation Extender for Integration Servers:

- WebSphere Transformation Extender for Message Broker
- WebSphere Transformation Extender for WebSphere Process Server and WebSphere ESB

The Eclipse-based Design Studio can be installed into an existing WebSphere Message Broker Toolkit or WebSphere Integration Developer installation, providing a single development tool for flow creation and data transformation.

WebSphere Transformation Extender for Message Broker

WebSphere Transformation Extender for Message Broker provides the capability to run WebSphere Transformation Extender transformation maps from within a WebSphere Message Broker message flow. This product enhances the enterprise service bus features of WebSphere Message Broker with the universal transformation capability of WebSphere Transformation Extender. Therefore, you can use this extender to transform and distribute different messages across various systems and applications.

WebSphere Transformation Extender for Message Broker complements the native message transformation and message parsing capabilities of WebSphere Message Broker by providing a plug-in node for running WebSphere Transformation Extender maps, called the *WTX Map node*. The WTX Map node is in the WebSphere Transformation Extender for Message Broker drawer of the WebSphere Message Broker Toolkit message flow node palette. You use the message flow node palette to add WTX Map nodes in the message flows that you are creating in the Message Flow editor. WebSphere Transformation Extender maps are deployed like any other Message Broker artifact as part of a broker archive (BAR) file.

WebSphere Transformation Extender adds support for custom formats that are difficult to support with the WebSphere Message Broker MRM (its basic message definition capability). It also provides rich industry support through the WebSphere Transformation Extender Industry Packs, providing the ability to handle complex and proprietary data types. Use of WebSphere Transformation Extender provides advanced data validation without complex coding. There is full support for multiple sources and multiple targets within a single transaction scope. In addition, it is fully compatible with Message Broker transaction control. WebSphere Transformation Extender for Message Broker takes full advantage of the scalability of WebSphere Message Broker.

WebSphere Transformation Extender for WebSphere Process Server

You use WebSphere Transformation Extender for WebSphere Process Server to enhance solutions that are built with the WebSphere ESB, WebSphere Process Server, and the WebSphere Integration Developer products. These products use default WebSphere Transformation Extender bindings that are ready to deploy WebSphere Transformation Extender in mediation flows and business processes. These default bindings can be generated from within WebSphere Integration Developer, the development studio for WebSphere Process Server and WebSphere ESB. WebSphere Transformation Extender for Integration Servers can also enhance solutions that are built with WebSphere Business Services Fabric.

When Design Studio is installed with WebSphere Integration Developer, an Eclipse-based wizard simplifies the map design. This edition takes advantage of the Service Component Architecture.

1.4.4 WebSphere Transformation Extender for Application Programming

WebSphere Transformation Extender for Application Programming is used to deploy transformations in custom programming environments and application servers. The C, Java, Enterprise JavaBeans™ (EJB™), COBOL and .Net application programming interfaces (APIs) are available to engage the transformation engine to process a pre-built transformation directly from the application.

1.4.5 WebSphere Transformation Extender SDK

The WebSphere Transformation Extender software development kit (SDK) delivers the *development* environment for all of the APIs such as C, Java, EJB, and COBOL to integrate custom programming environments directly to WebSphere Transformation Extender. In addition, you can develop custom adapters for WebSphere Transformation Extender, by using the SDK, and deploy them in any base edition. The SDK is available for writing client programs that connect directly to the stand-alone WebSphere Transformation Extender transformation engine. It also provides sample programs that show how you can call the WebSphere Transformation Extender engine from several different APIs. The WebSphere Transformation Extender SDK adds the capability of the Application Programming edition to the other base editions.

1.5 WebSphere Transformation Extender on System z

Mainframes continue to play a key role in many enterprises, both in traditional and SOA environments. A single mainframe can perform the work of several servers, reducing administration overhead and the green footprint from an energy-conserving perspective. In some cases, the volume of data is simply too large to be moved off the mainframe (for example, the data in a very large database). In other cases, a system does not provide an upgrade path.

Data affinity is a common justification for transforming data on the mainframe itself, where the critical business applications reside. Also, mainframes can simply be the most-reliable and scalable platforms upon which to run corporate data for day-to-day business functions.

Many enterprises have made strategic commitments to deploy data-integration applications on mainframes. These enterprises want to continue taking advantage of the scale, security, and expertise that they have on the mainframe.

The WebSphere Transformation Extender engine is available on the System z platform for embedding in COBOL or PL/I applications, batch processing. It can also be integrated with IBM DB2® software and participate in IBM CICS or IBM IMS transactions. WebSphere Transformation Extender also supports Linux on System z technology.

WebSphere Transformation Extender on System z provides the ability to quickly connect and interface existing batch applications among themselves and with industry standards. By using Design Studio on Windows, users build processing and integration rules, integrate data of disparate types from disparate sources, and process data objects natively without the need to program in COBOL, PL/I or Java. WebSphere Transformation Extender can work directly in any System z architecture.

WebSphere Transformation Extender development is the same for all platforms. When deploying a map, you use Design Studio to build the map for a specific platform, including the z/OS and Linux on System z platforms, as shown in Figure 1-2.

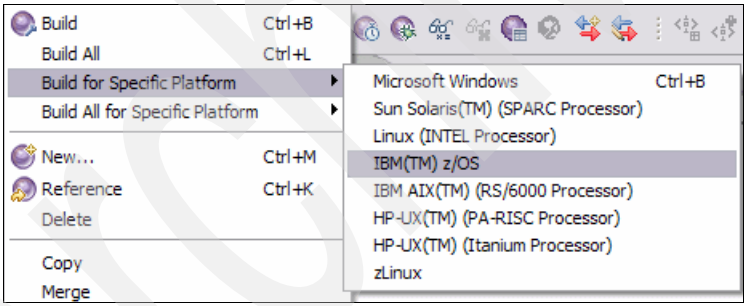

Figure 1-2 Build for specific platform

WebSphere Transformation Extender also provides WebSphere MQ connectivity so that batch data transformations can rapidly use WebSphere MQ for source or target applications. The WebSphere Transformation Extender MQ connector provides a “plug-and-play” bridge between existing file-based batch applications and WebSphere MQ-enabled applications. It seamlessly integrates into existing batch operations without changes to current monitoring and management procedures.

WebSphere Transformation Extender on System z provides the following advantages among others:

- Eliminates custom integration programming for transforming data
- Avoids costly changes to applications written to older industry standards
- Improves quality assurance by validating information exchanges outside of the applications
- Reduces time and cost, maintaining interfaces to evolving standards
- Allows for fewer IT backlogs and code free maintenance

1.5.1 WebSphere Transformation Extender with Command Server on System z

With WebSphere Transformation Extender with Command Server on z/OS, the transformation engine can be called directly from a command line, JCL, or timer. WebSphere Transformation Extender Command Server is a powerful command-line-driven environment to control the execution of a transformation. Command Server can be used to extend an existing application and infrastructure with minimal intrusion.

Users of MVS Batch, CICS, and IMS environments on System z use the Command Server edition. For System z users, this edition includes the components in the Application Programming edition. Command Server on z/OS supports the following environments:

- Batch or JCL
- CICS
- IMS

1.5.2 WebSphere Transformation Extender with Launcher on System z

WebSphere Transformation Extender with Launcher runs under UNIX System Services on System z. WebSphere Transformation Extender with Launcher includes the WebSphere Transformation Extender Launcher and the Command Server.

The Launcher activates transformations when triggered by an event. The event might be a file creation, a message arriving on a WebSphere MQ queue, a database trigger, a scheduler, and many other options including combinations of triggers. The Launcher offers an online run-time environment that synchronizes and controls complicated data transformations while maintaining and optimizing the execution environment for the transformations. WebSphere Transformation

Extender with Launcher provides an event-driven (even on native files), multi-threaded, multi-process transformation engine.

1.5.3 WebSphere Transformation Extender for Integration Servers on System z

WebSphere Transformation Extender for Integration Servers provides extenders to other WebSphere products, providing installation and integration support. WebSphere Transformation Extender complements the native mediation and business-rule capabilities of these products. It can also process large documents and messages with more complex formats that are not based on XML. It allows you to solve critical business problems in real time with support for industry standards and regulatory compliance.

The following WebSphere Transformation Extender products are packaged as WebSphere Transformation Extender for Integration Servers on System z:

- WebSphere Transformation Extender for Message Broker on z/OS
- WebSphere Transformation Extender for WebSphere Process Server and WebSphere ESB

1.5.4 WebSphere Transformation Extender for Application Programming on System z

WebSphere Transformation Extender for Application Programming is used to deploy the transformation engine in custom applications and application servers, and use the APIs provided to execute a transformation map. You use WebSphere Transformation Extender for Application Programming when you want to call directly from your application into the transformation engine. WebSphere Transformation Extender for Application Programming on System z provides APIs and interfaces for the following application types:

- C
- COBOL
- Enterprise JavaBeans (EJB)
- Java
- PL/I

1.5.5 WebSphere Transformation Extender on Linux on System z

The concept of virtualization where large numbers of virtual servers are consolidated onto a small number of relatively large, physical servers is rapidly becoming commonplace in data centers around the world. The System z

mainframe is ideally suited for this large, hosting server role. WebSphere Transformation Extender offers 64-bit operating system tolerance support for selected platforms and availability on Linux on System z, including the following platforms:

- RedHat Linux on System z
- SUSE® Linux on System z

WebSphere Transformation Extender installs and operates on Linux on System z just as on Linux on distributed systems. The following extenders are available on Linux on System z:

- WebSphere Transformation Extender with Command Server
- WebSphere Transformation Extender with Launcher
- WebSphere Transformation Extender for Integration Servers
- WebSphere Transformation Extender for Application Programming
- WebSphere Transformation Extender SDK

1.6 WebSphere Transformation Extender with other products

WebSphere Transformation Extender can also be used with some other products in the IBM software portfolio, including Lotus Forms and IBM InfoSphere™ Information Server.

1.6.1 Lotus Forms

With the Lotus Forms suite of products, organizations can use electronic forms to gather information from users and transmit that information to other systems. Lotus Forms can be used as the front end for any business process, such as opening a new account. When a customer enters their information into a form and submits it for processing, their information can pass into a workflow application (such as WebSphere Process Server), a database (such as DB2 Universal Database™ or DB2 Content Manager), or any other type of application or process.

You can extend a Lotus Forms system by using WebSphere Transformation Extender. With the Forms Services Platform's Designer plug-ins, you can generate WebSphere Transformation Extender maps from form instances and schemas. A WebSphere Transformation Extender sample is provided to show how you can use WebSphere Transformation Extender in Forms pipelines, as a step in the form processing. The sample uses WebSphere Transformation Extender to extract and map a form data to an HTML response page.

By combining WebSphere Transformation Extender with Lotus Forms, a developer can easily use all of the industry standards that are provided by the WebSphere Transformation Extender Industry Packs and Enterprise Packs. See Chapter 2, “Overview of the industry solutions for WebSphere Transformation Extender” on page 21. With these packs, a developer has the ability to handle such structures as XML, EDI, and HIPAA documents.

1.6.2 IBM InfoSphere Information Server

IBM InfoSphere Information Server is a revolutionary software platform that helps organizations derive more value from the complex, heterogeneous information that is spread across their systems. It provides breakthrough collaboration, productivity, and performance for cleansing, transforming, and moving this information consistently and securely throughout the enterprise. The information can then be accessed and used in new ways to drive innovation, increase operational efficiency, and lower risk.

IBM InfoSphere Information Server includes several products, such as WebSphere DataStage® and QualityStage. These products help companies integrate information to deliver faster and higher quality business results on the following types of business initiatives:

- Business intelligence
- Master data management
- Infrastructure rationalization
- Business transformation
- Risk and compliance

WebSphere Transformation Extender for DataStage (also known as *MapStage*) provides advanced transformation power to IBM InfoSphere DataStage and QualityStage. This power includes parsing, validation, and transformation of complex hierarchical file structures, such as XML, EDI, SWIFT, and HIPAA documents, so that they can be more easily processed by IBM InfoSphere Information Server.

WebSphere Transformation Extender for DataStage has the following functionality and benefits:

- Executes existing WebSphere Transformation Extender maps within DataStage jobs
- Supports complex metadata formats supported only by WebSphere Transformation Extender type trees, for example, XML, EDI, SWIFT, and so on
- Supports the creation of WebSphere Transformation Extender type trees and maps based on the selected input and output link metadata

- Supports the editing and configuration of the WebSphere Transformation Extender map and card options
- Provides access to WebSphere Transformation Extender connectivity adapters

2

Overview of the industry solutions for WebSphere Transformation Extender

In this chapter, we describe industry-specific WebSphere Transformation Extender Packs that are available to help you develop and maintain solutions based on industry-standard, information-exchange formats. We also describe the industry packs and enterprise packs and provide information about some of the adapters that are used by WebSphere Transformation Extender. In addition, we discuss WebSphere Transformation Extender Trading Manager.

2.1 Industry packs

WebSphere Transformation Extender Industry Packs help accelerate time to value as you develop and maintain solutions based on industry-standard, information exchange formats that you need to transform into internal business-application formats. Industry packs provide ready-to-use capabilities to integrate with a range of industry standard data formats with your enterprise infrastructure. Industry packs provide predefined type tree templates, conversion maps, and in selected packs, provide validation maps and tools to reduce risk from projects that require conformance to mandatory and advisory guidelines required by regulatory bodies or industry service providers.

When used with WebSphere Transformation Extender, WebSphere Transformation Extender Industry Packs provide an infrastructure that offers the following advantages:

- Enables compliance with government or industry mandates
- Controls administrative costs
- Streamlines business processes
- Facilitates the accuracy and timeliness of information
- Reuses existing business systems
- Adapts to new technologies as they emerge
- Integrates multiple systems and standards
- Automates cross-organizational exchanges
- Delivers trustworthy information for critical business initiatives

Industry packs are combined into industry groups for the purpose of announcements of new releases that are related to the packs, and pack versions are aligned to the group versions. WebSphere Transformation Extender offers the following industry-specific packs, each of which are explained in detail in the sections that follow:

- WebSphere Transformation Extender Packs for Healthcare
  - HIPAA electronic data interchange (EDI)
  - Health Level Seven (HL7)
  - National Council for Prescription Drug Programs (NCPDP)
- WebSphere Transformation Extender Packs for Financial Services
  - National Automated Clearing House Association (NACHA)
  - Single Euro Payments Area (SEPA)
  - Financial eXchange™ Protocol (FIX)
  - SWIFTNet FIN
  - SWIFTNet Funds

- WebSphere Transformation Extender Packs for Insurance
  - Association for Cooperative Operations Research and Development (ACORD)
- WebSphere Transformation Extender Packs for EDI
  - WebSphere Transformation Extender Pack for EDI for Administration, Commerce, and Transport (EDIFACT)
  - WebSphere Transformation Extender Pack for X12
  - WebSphere Transformation Extender Pack for EANCOM
  - WebSphere Transformation Extender Pack for Trading Data Communications Standard (TRADACOMS)
  - WebSphere Transformation Extender Pack for Odette

As standards evolve and are updated, IBM provides timely updates to the pack contents, helping to protect existing investments year after year. Pack contents are imported into Design Studio tools to help ensure that your business can quickly assimilate new industry-led requirements.

2.1.1 Healthcare

WebSphere Transformation Extender offers several industry packs for healthcare as described in the following sections.

WebSphere Transformation Extender Pack for HIPAA EDI

WebSphere Transformation Extender Pack for HIPAA EDI includes ready-to-execute template definitions for the complete ASC X12N standard for HIPAA. It includes the HIPAA 4010 addendum. It also includes the CMS (formerly HCFA) formats for legacy NSF, UB-92 Claims and Coordination of Benefits, and 4010 flat file formats for Professional and Institutional Claims and Coordination of Benefits.

This pack includes support for HIPAA X12 5010-based standards for HIPAA transactions. Support for these standards is critical because they will be mandated, in the future, for use, and the 4010 addenda (4010a1) version of the HIPAA transactions will no longer be allowed. Therefore, this pack supports both 4010A1- and 5010-based HIPAA standards and allows you to transition from a 4010A1-only environment to a mixed and eventually a 5010 environment as easily as possible.

The pack features compliance validation checking and handling of claims attachments. The pack also supports the National Provider ID requirement and Clinical Data Architecture release 2.0 for claims attachments support.

WebSphere Transformation Extender Pack for HL7

WebSphere Transformation Extender Pack for HL7 includes HL7 versions 2.1, 2.2, 2.3, 2.4, and 2.5 type trees. HL7 version 3 Extensible Markup Language (XML) formats can be easily imported into the Design Studio Type Tree Editor by using the XML Importer that is built into the tool.

WebSphere Transformation Extender Pack for NCPDP

WebSphere Transformation Extender Pack for NCPDP includes template definitions for all of the finalized HIPAA NCPDP Telecommunications V5.1 transaction sets and the Batch V1.1 format. The pack also supports NCPDP V3.2 transaction sets and the Batch V1.0 format. Sample maps are included to create reports from requests and replies.

2.1.2 Financial services

WebSphere Transformation Extender offers several industry packs for financial services as described in the following sections.

WebSphere Transformation Extender Pack for NACHA

WebSphere Transformation Extender Pack for NACHA delivers templates for the 2007 Automated Clearing House (ACH) compliant message exchange. The pack provides extensive compliance by using validation rules and content restrictions that are included in the type tree definitions. Type trees are provided that represent the technical specifications of the ACH rules for 2007, 2008, and 2009. Maps are included for conversion of Cash Concentration and Dispersion (CCD) and Corporate Trading Exchange (CTX), and International ACH Transfer (IAT) ACH formats.

WebSphere Extender Pack for SEPA

WebSphere Transformation Extender Pack for SEPA helps banks and financial institutions meet compliance obligations to the European Union initiative to have a pan-European electronic payments infrastructure. The pack includes templates for SEPA credit transfers and direct debits, mapping between SEPA and SWIFT message type103, and validation for European Payment Council's SEPA implementation guidelines. Additional converter examples for mapping MT101, 102, 102+, and 103+ SWIFT message types to SEPA credit transfer messages are included. Additional validation updates are implemented from the European Payment Council updated guidelines of June 2007. This pack also provides sample maps between some of the European domestic formats and SEPA formats.

The SEPA pack also includes support for the European Payments Council (EPC) Technical Validation Subset (TVS) schemas and Euro Banking Association

(EBA) STEP2 schemas for the relevant payments clearing and settlement messages. Type trees generated for these with the WebSphere Transformation Extender Schema Importer are included. Additional validation maps are also provided that are used to validate the general usage rules as defined in the EPC rule books and the EBA STEP2 rule book.

WebSphere Transformation Extender Pack for SWIFTNet FIN

WebSphere Transformation Extender Pack for SWIFTNet FIN provides support for SWIFTNet FIN SRG 2007 (ISO 7775 and 15022). In addition to templates and maps, it includes two levels of validations (type trees and a Java Validation Component) to ensure message conformity toward Society for the Worldwide Interbank Financial Telecommunication (SWIFT) message standards. It also delivers monitoring and prioritization of message storage and retrieval. Samples are included for Swift Alliance Access (SAA) headers for SWIFTNet Phase 2.

WebSphere Transformation Extender Pack for SWIFTNet Funds

WebSphere Transformation Extender Pack for SWIFTNet Funds provides support for ISO20020 message sets and transformations between SWIFTNet Funds and SWIFT Fin SRG 2006. In addition, it provides enhanced network validation for messages and support for the additional messages that are added in SWIFTNet Fund version 3.0 and 3.1.

WebSphere Transformation Extender Pack for FIX

WebSphere Transformation Extender Pack for FIX delivers prebuilt data transformation content for the Financial Information Exchange (FIX) Protocol. FIX is a set of message specifications for the automated trading of financial instruments and is used as a standard format in straight-through-processing and collaboration scenarios by banks, brokers, dealers, and market makers, including the foreign exchange markets. FIX formats are available in both older tag=value and XML (FIXML), which are both document type definition (DTD) and XML schema based. The pack supports versions of both formats.

2.1.3 Insurance

WebSphere Transformation Extender offers an industry pack for insurance as described in the following section.

WebSphere Extender Pack for ACORD

WebSphere Extender Pack for ACORD provides prebuilt message templates for the ACORD insurance industry standards for Life & Annuity (Life) V2.x and Property & Casualty/Surety (PCS) V1.x. An AL3 2006 V4 type tree is included

with a conversion map example for AL3 to the ACORD PCS XML 1.x format by using the ACORD PCS V1.10.1 schema. A validation map utility with XML data examples for PCS is provided. Life support includes a sample map for TXLife 2.15.02 to X12 EDI 267 (individual life annuity and disability application). This pack requires the ACORD XML schema, which is available to ACORD members only.

2.1.4 EDI

WebSphere Transformation Extender offers several industry packs for EDI as described in the following sections.

WebSphere Transformation Extender Pack for EDIFACT

WebSphere Transformation Extender Pack for EDIFACT includes prebuilt definitions for the United Nations EDIFACT standards.

WebSphere Transformation Extender Pack for X12

WebSphere Transformation Extender Pack for X12 includes prebuilt definitions for the ASC X12 transaction sets. ASC X12 represents multiple business domains including finance, government, health care, insurance, and transportation.

WebSphere Transformation Extender Pack for EANCOM

WebSphere Transformation Extender Pack for EANCOM provides prebuilt definitions for EDI solutions requiring standard exchange formats. The EAN identification system defines unique identification codes for products and locations that are often used for barcodes.

WebSphere Transformation Extender Pack for TRADACOMS

WebSphere Transformation Extender Pack for TRADACOMS provides prebuilt definitions of TRADACOMS formats.

WebSphere Transformation Extender Pack for Odette

WebSphere Transformation Extender Pack for Odette provides a template to help improve quality and shorten the design time of EDI solutions for the automotive industry.

2.2 Enterprise packs

WebSphere Transformation Extender provides a range of technology connectors and adapters that are delivered as a single package. WebSphere Transformation Extender Enterprise Packs provide ready-to-use capabilities to accelerate the connectivity of WebSphere Transformation Extender with key independent software vendor (ISV) business applications. The following enterprise packs are offered:

- WebSphere Transformation Extender for PeopleSoft®
- WebSphere Transformation Extender for SAP R/3®
- WebSphere Transformation Extender for SAP XI
- WebSphere Transformation Extender for Siebel®

Each Enterprise Pack provides the capability for WebSphere Transformation Extender to use one or more interfaces that are provided by the vendor for external applications to communicate and exchange business information.

In addition to detailed documentation of the interfaces and the steps needed to integrate the application, adapters and plug-in “assist” applications are provided for both synchronous and asynchronous connectivity to the WebSphere Transformation Extender transformation engine. Where the ISV application supports and publishes well defined data exchange formats (for example SAP), importers are included that help discover and replicate the application-specific metadata. These importers ensure a higher degree of compliance with the syntax, semantics, and structure. By using these enterprise packs, developers, testers, and production staff can accomplish more through deeper integration with their enterprise environments.

2.2.1 WebSphere Transformation Extender for PeopleSoft

The WebSphere Transformation Extender Pack for PeopleSoft accelerates the real-time integration and reuse of PeopleSoft business applications with other e-commerce, customer relationship management (CRM), supply chain, enterprise resource planning (ERP), custom, and existing applications. The pack describes PeopleSoft integration methods. It provides examples that show how to incorporate the methods into your enterprise by using the WebSphere Transformation Extender Design Studio Importer Wizard, Type Tree Editor, Map Editor, and Integration Flow Designer.

The following PeopleSoft integration interfaces, among others, are supported by the WebSphere Transformation Extender Pack for PeopleSoft:

Application Messaging
: An application integration feature based on subscribing to and publishing XML formatted messages.

Component Interfaces
: An application integration interface that provides access to the PeopleSoft business rules and data that is associated with the components.

The pack contains the following components, among others, to facilitate the development of PeopleSoft interfaces for use with WebSphere Transformation Extender:

- Component Interface Adapter
- Component Interface Importer
- Templates
- Examples

This pack includes the WebSphere Adapter for PeopleSoft Enterprise V6. The PeopleSoft adapter adds a listener interface for PeopleSoft Component Interfaces.

2.2.2 WebSphere Transformation Extender for SAP R/3

WebSphere Transformation Extender Pack for SAP R/3 accelerates the real-time integration and reuse of SAP business applications with other e-commerce, CRM, supply chain, ERP, custom, and existing applications. The universal transformation capabilities of WebSphere Transformation Extender, together with the Pack for SAP R/3, make communications with and from other business applications seem as though they are another SAP R/3 system.

The pack uses SAP's synchronous and asynchronous (transactional) Remote Function Calls (RFCs). The pack removes the need for detailed knowledge of RFCs to create interfaces for SAP R/3 that are reliable, efficient, and easy to maintain. In addition, the pack can be used with several file-based interfaces, such as SAP DXOB/DMI and EDI subsystems. The pack also contains a custom Batch Data Communications (BDC) solution that makes it possible to use BDC from either a file or an RFC-based interface.

The WebSphere Transformation Extender Pack for SAP R/3 supports the following interfaces and technologies that include the automatic generation of data-specific type trees:

- Batch Data Communication (BDC)
- Business Application Programming Interface (BAPI®, JBAPI)
- DXOB/Data Migration Interface (DMI)
- Intermediate Documents (IDOCs)
  - Application Link Enabling (ALE, JALE)
  - ALE Message Handler (AMS)
  - Electronic Data Interchange (EDI)

The pack contains the following components, among others, to facilitate the development of SAP R/3 interfaces for use with WebSphere Transformation Extender:

- ALE Adapter
- BAPI Adapter
- BAPI Importer
- BDC Adapter
- BDC Importer
- BW Staging BAPI
- BW OLAP BAPI
- Correction and Transport files for BDC
- DXOB Importer
- Examples for each of the SAP R/3 interfaces supported
- IDOC Importer

This pack includes the WebSphere Adapter for SAP Software V6. The SAP Adapter adds support for the Query Interface for SAP Software (QISS).

2.2.3 WebSphere Transformation Extender for SAP XI

The WebSphere Transformation Extender Pack for SAP XI accelerates the real-time integration and reuse of SAP business applications with other e-commerce, CRM, supply chain, ERP, custom, and existing applications, where WebSphere Transformation Extender connects into the SAP Exchange Infrastructure (XI).

The SAP XI Pack is unique because it supports the WebSphere Transformation Extender engine as an embedded component of the SAP Web server. This support satisfies all of the SAP user's transformation and validation needs when integrating applications other than SAP into an environment that is primarily SAP.

The pack contains the following components, among others, to facilitate the development of SAP XI interfaces for use with WebSphere Transformation Extender:

- ALE Adapter
- Examples
- IDOC Importer
- Transformation Plug-in
- User interface to build configuration files

2.2.4 WebSphere Transformation Extender for Siebel

The WebSphere Transformation Extender Pack for Siebel accelerates the real-time integration and reuse of Siebel business applications with other e-commerce, CRM, supply chain, ERP, custom, and existing applications. With the WebSphere Transformation Extender Pack for Siebel, you can use the following Siebel integration methods:

- Workflow Process Manager

  Siebel's Workflow Process Manager is an XML messaging interface. It is used to integrate, share, and replicate data between a Siebel system and an external application. The WebSphere Transformation Extender Pack for Siebel provides a custom Siebel business service and plug-in. This plug-in that handles the WebSphere Transformation Extender map execution process from within a Siebel workflow process with synchronous and asynchronous communication support for external applications.

- Siebel Data Bean

  The Siebel Data Bean provides a robust, business object interface between Siebel applications and external applications. In this scenario, a WebSphere Transformation Extender with Launcher process can be used to access, convert and transmit data from the Siebel application by using the Business Object adapter that is provided in the pack. The adapter uses technology that allows the WebSphere Transformation Extender user to access the Siebel application for reading data without having to display any user interface.

The pack contains the following components, among others, that facilitate the development of Siebel interfaces for use with WebSphere Transformation Extender:

- Business Services
- Business Service Plug-In
- Examples for each of the Siebel interfaces supported
- Siebel Business Object Adapter
- Siebel Com Business Object Adapter
- Siebel Com Business Object Importer

This pack includes the WebSphere Adapter for Siebel V6. The adapter adds a listener interface for Component Interfaces.

2.3 Adapters

WebSphere Transformation Extender provides resource adapters that are used to retrieve and route data. They provide access to databases, files, messaging systems, enterprise applications, and other data sources and targets. An adapter is a technical connector that can be “plugged” into a source or destination, without parsing. You can use adapters with the Command Server, Launcher, or with a map in a map rule. A subset of adapters is supported on z/OS and includes WebSphere MQ, file, and FTP.

Each adapter includes a set of adapter commands that can be used to customize its operation. For example, when using the WebSphere MQ adapter, you use the adapter commands to specify different queues and queue managers, specific messages by message ID, specific sets of messages with the same message ID, message descriptors in your data, and more.

Resource adapters answer the questions “Where should the data come from?” and “Where should the data go?”. The resource adapters that are provided with the WebSphere Transformation Extender products enable data transformation and adapter-specific behavior recognition on different systems and data structures.

Source and target data can be in a file, from a specific application, in a database, from a message queue, from an archive file, or from other sources. The client components of the Design Studio, Integration Flow Designer, Map Editor, Type Tree Editor, and Database Interface Designer are integral to using the resource adapters.

Some of the adapters (partial list) that are provided include support for the following data sources and targets:

- Base64
- CICS
- COM
- CORBA
- DB2
- File
- HTTP
- Informix®
- Java class
- JMS

- JNDI
- Microsoft SQL Server®
- MIME
- ODBC
- OLE DB
- Oracle®
- SNMP
- SOAP
- Sybase
- WebSphere MQ

2.3.1 JCA adapters

A group of WebSphere Adapters is also provided. The WebSphere Adapters are Java EE Connector Architecture (JCA) based. The following adapters are specifically included:

E-mail	Includes support for multiple attachments and headers
FTP	Allows remote files to be renamed after they have been read
SAP	Adds support for the Query Interface for SAP Software
PeopleSoft	Adds a listener interface for Component Interfaces
Siebel	Adds a listener interface for Component Interfaces

Note: The SAP, PeopleSoft, and Siebel adapters are available only in the Enterprise Packs.

Each WebSphere Adapter includes a new Enterprise Metadata Discovery (EMD) importer, which can be used to generate the following information:

- XML schemas (in lieu of type trees)
- Service definitions (similar to MDQ files)
- Adapter command lines

At the point where the adapter is bound to the card, a Command Line Wizard is available.

2.3.2 Adapters on System z

When using WebSphere Transformation Extender for z/OS in batch, CICS, and IMS, the following adapters are provided:

- DB2
- Files (QSAM and VSAM)
- FTP
- MQ

When using WebSphere Transformation Extender with Launcher under UNIX System Services, the following adapters are provided:

- CICS
- DB2 (MVS and CLI)
- EMAIL
- Files (hierarchical and native file systems)
- FTP
- MQ
- Oracle

2.4 WebSphere Transformation Extender Extended Functions

A couple separate extended functions are available to provide specialized functionality. These functions include the WebSphere Transformation Extender Secure Adapter Collection and the WebSphere Transformation Extender SNMP Collection.

2.4.1 IBM WebSphere Transformation Extender Secure Adapter Collection

The WebSphere Transformation Extender Secure Adapter Collection provides additional technology adapters to encrypt and decrypt data by using several popular standards, including Secure Multipurpose Internet Mail Extensions (S/MIME), Secure Sockets Layer (SSL), and OpenPGP. WebSphere Transformation Extender Secure Adapter Collection works with any base edition.

2.4.2 WebSphere Transformation Extender SNMP Collection

The WebSphere Transformation Extender SNMP Collection enables integration with network management applications. The Simple Network Management Protocol (SNMP) Collection includes an SNMP Agent. The SNMP Agent enables monitoring and management by an SNMP manager and an SNMP Adapter. The adapter enables the sending of alerts to SNMP manager products from WebSphere Transformation Extender maps.

2.5 Trading Manager

Electronic commerce data flows from external sources to internal destinations and from internal sources to external destinations, and possibly to other internal organizations. *WebSphere Transformation Extender Trading Manager* is a client/server product for managing and processing electronic commerce data that provides the capabilities for managing and controlling the business-to-business integration of partner relationships and message flow. Trading Manager users can audit, control, monitor, and view the entire business-to-business integration environment across the extended enterprise with secure data exchange fully integrated with back-end systems.

WebSphere Transformation Extender Trading Manager is an application of WebSphere Transformation Extender with Launcher deployed with EDI Industry Packs as a business-to-business hub. WebSphere Transformation Extender Trading Manager consists of two components:

- Partner Manager

  This component is a graphical application that is used to administer partner and trade link information, create a comprehensive database of an organization's electronic commerce information, and manage e-commerce activity. You use Partner Manager to perform the following tasks:

  - Define the trading relationships with EDI partners, including details such as the version of each document, and control numbering schemes, tracking, error handling, status, and the post office used.
  - Maintain the information for each trading partner, including details regarding addresses, contacts, and the individual departments or other groups in trading-partner organizations.
  - Provide audit and control information that is collected in the Trading Manager database that is used to monitor ongoing inbound and outbound e-commerce transaction activity.
  - Provide alerts when error conditions occur in e-commerce inbound or outbound processing.

- Message Manager

  This component is a runtime system that manages the integration of partner messages by using a predefined system of maps that process electronic commerce data. You use these maps to validate and route data. The Message Manager is the runtime component of the Trading Manager.

Message Manager supports two distinct data flows:

– Inbound data flow

  This flow typically represents the receipt of data in an EDI or proprietary format from an external trading partner.

– Outbound data flow

  Typically this flow represents the retrieval of data from an internal application system in its native format for subsequent conversion and routing to an external trading partner.

The information from Partner Manager is shared with Message Manager for validating, tracking, and routing e-commerce data. Information about this e-commerce activity is then incorporated into the Partner Manager database for monitoring and reporting.

WebSphere Transformation Extender Trading Manager complements WebSphere Partner Gateway. WebSphere Partner Gateway provides AS1, AS2, and AS3 channel support for EDI users who are transitioning from value added networks (VANs) to Internet EDI. Trading Manager also complements WebSphere DataPower XB60, which is a purpose-built business-to-business hardware appliance for simplified deployment, exceptional performance, and hardened security with support for AS2 and AS3.

Part 2

WebSphere Transformation Extender concepts and functionality

In this part, we discuss the basic concepts of WebSphere Transformation Extender that you must understand to build, deploy, and manage transformations. We then review the various methods of execution of a transformation within each of the editions of WebSphere Transformation Extender.

This part includes the following chapters:

- Chapter 3, “Basic concepts of WebSphere Transformation Extender” on page 39
- Chapter 4, “Methods of execution” on page 93

3

Basic concepts of WebSphere Transformation Extender

In this chapter, we provide details about the basic concepts of WebSphere Transformation Extender:

- Design process
- Type trees
- Maps
- Systems

Knowledge of these concepts is required to understand how to use WebSphere Transformation Extender to build, deploy, and manage transformations.

3.1 Design process

Modeling any data transformation in WebSphere Transformation Extender requires three steps. Figure 3-1 shows the steps to create a Transformation Extender solution.

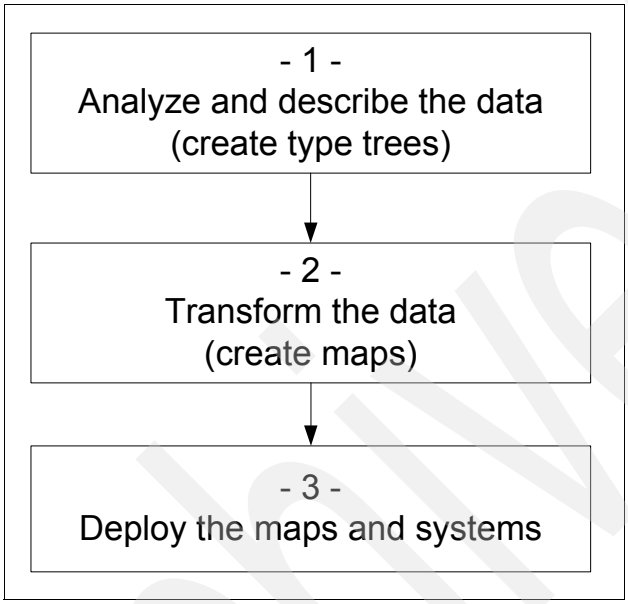

Figure 3-1 The steps to create a Transformation Extender solution

We describe each of these steps as follows:

1. Analyze and describe the data.

   In this step, you create *type trees,* which tell WebSphere Transformation Extender how the data is structured. You must understand the input and output data. In addition, you must perform the following tasks:

   – Identify the type of data in order to determine whether type trees can be generated by using an importer or the Database Interface Designer, or if you can use a predefined type tree.

   – If type trees cannot be generated, use the Design Studio to create type trees to describe properties and structures of input and output data.

2. Transform the data.

   You must create *maps* that specify data transformation logic to be used when map is run.

3. Deploy maps and systems.

   You must perform the following actions:

   – Link systems of maps together. If you are using WebSphere Transformation Extender with Launcher, you must create *systems* that link the maps and create a transformation workflow. For other runtime environments, it is enough to build and deploy the maps.

   – If your system is destined for the Launcher runtime, you must specify specific events that should start the execution of transformation workflows. The events can be based on incoming source data, or they can be time based. You then assign the system to a destination server.

   – Deploy completed maps and systems to production servers.

You use WebSphere Transformation Extender Design Studio to create the type trees and maps. Systems are created in the Integration Flow Designer. All database objects' management and settings can be done with the Database Interface Designer, including the generation of type trees.

Type trees can be generated automatically through a wizard for several formats, such as database tables, XML DTDs, enterprise resource planning (ERP) systems, COBOL copybooks, and more. The type trees support grouping, which enables the handling of "nested levels," as in SWIFT electronic data interchange (EDI) and XML. The type trees also handle validation of data, both input and output, through rules that are associated with the data definitions. We explain the basic concepts of type trees in 3.2, "Type trees" on page 42.

After the generation or creation of the type trees, you validate the type tree. You can do this by using a validation map. See 3.3.5, "Validation maps" on page 84.

After your type trees are validated, you can create the maps that transform the data. You start by adding input and output cards to the transformation map, at least one card per physical input and output datastream. You define metadata as being the schema of the card, in which you declare the type tree that represents the metadata of each card and the level within the type tree that is used to describe the data. WebSphere Transformation Extender can handle multiple inputs or multiple outputs (many-to-many, any-to-any). The mapping is a simple process to describe how to generate output data. For each field of each output card, rules can be added in which you can use a simple drag-and-drop method for the types from input cards, mathematical formulas, conversions, literal values, or a concatenation of any or all of these. We describe the basic concepts of maps in 3.3, "Maps" on page 72.

After you define your mappings, you can build (compile) a pseudo-executable file that contains all your transformation logic and is interpreted by the transformation server. Compilation must be done specifically for the target platform.

Depending on the target environment, you can use the compiled maps directly, or you might need to create a system. A *system* is a set of logically related maps or subsystems that represent a transformation unit. The system diagrams show the workflow with its data sources, components, references, servers, and data targets. The Integration Flow Designer is used during development to define and manage the system. We describe the basic concepts of systems in 3.4, “Systems” on page 86.

3.2 Type trees

A *type tree* is a collection of data definition properties and characteristics. Type trees have the .mtt file name extension. Each icon in a type tree identifies a data type. The tree has a root type, and other types are connected to the tree through branches.

Similar to a data dictionary, the type tree view is a list of all the objects that form the data. The type tree view is a classification hierarchy. As you go to deeper levels of the tree, you move to a more specific definition. Think of the nested levels as file folders that are used to organize the objects that form the data.

Like a dictionary, the types in a type tree are shown (by default) in alphabetic order. The sequence does not indicate anything about the order of the fields in the data itself.

The type trees are manipulated in the Type Tree Editor in the Design Studio. The data content in a type tree is not identified as the source or the target data. Instead, a type tree defines only the specific properties of the data, while the input and output destinations of the data are specified in the map.

A type tree supports data definitions for text or binary data and a variety of character sets. Predefined type trees are provided in add-on Integration Packs for many business integration standards. Such standards include those for EDI, such as ANSI X12, EDIFACT and TRADACOMS, Health Level7 (HL7), SWIFT, and more.

The Design Studio includes an importer tool for automatically generating type trees for data that is described in XML document type definitions (DTDs) and schemas, COBOL, SAP R/3, PeopleSoft, Java classes, and more.

Additionally, the Design Studio has a utility called the *Type Tree Maker*. This utility reads-in an XML document file (has the .mts file extension) that contains type tree commands and uses these commands to create or update a type tree.

You can also create a type tree manually. See 3.2.1, “Exploring type trees” in which we explain the steps to manually create a type tree.

3.2.1 Exploring type trees

If the type tree that is necessary to describe the data is unavailable as an industry pack, and it cannot be generated automatically by an importer, you must create it manually:

1. Describe the data.
2. Assign each element to a class.
3. Define the properties of the elements.
4. Define the components order in the data stream.
5. Analyze and save the type tree.

For a type tree that is imported, you start to work with the type tree somewhere in the middle of this flow. In most cases, you only need to do the last step, which is to analyze and save the type tree.

In this chapter, we present the concepts that you need to understand when building a type tree. For details about how to effectively build the type tree in Design Studio, see Chapter 5, “Developing in Design Studio” on page 171.

Describing the data

A key concept when using Design Studio is the idea of data objects. A *data object* is something in your data that you think of as a complete unit. It might be as simple as an employee hire date (a field) or as complex as an employee table made up of many rows and columns.

When defining a data object, you need to know the following information:

- What the data looks like
- If the data is a complex object, how to distinguish one object from the next
- If the data is a simple object, what value to assign to the data
- Constraints or context-specific limitations on valid data

Remember that the structure of the data must be identified correctly, so that when you run the map, it can parse the data upon input or build the data upon output. Also, some data can be defined in several ways, as long as your data definition matches the physical stream. For example, you can say that your input data is a file, without being more precise on the content. You can also say that the same input data is a file, made of records, each of which contains, for example, seven fields.

In your type tree, you create a type for each object. When defining each type, you define the properties and characteristics of each data object. Each object defined in the type tree is assigned to one of three classes: item, group, or category.

In a type tree, the smallest unit of data is an *item*. An item is indicated by the icon in the type tree. Types whose objects are made up of other objects are ultimately broken down into items. For example, in a file, the smallest unit of data is a *field*. A field is classified as an item. Most fields or data elements are items. Figure 3-2 shows examples of items in a file.

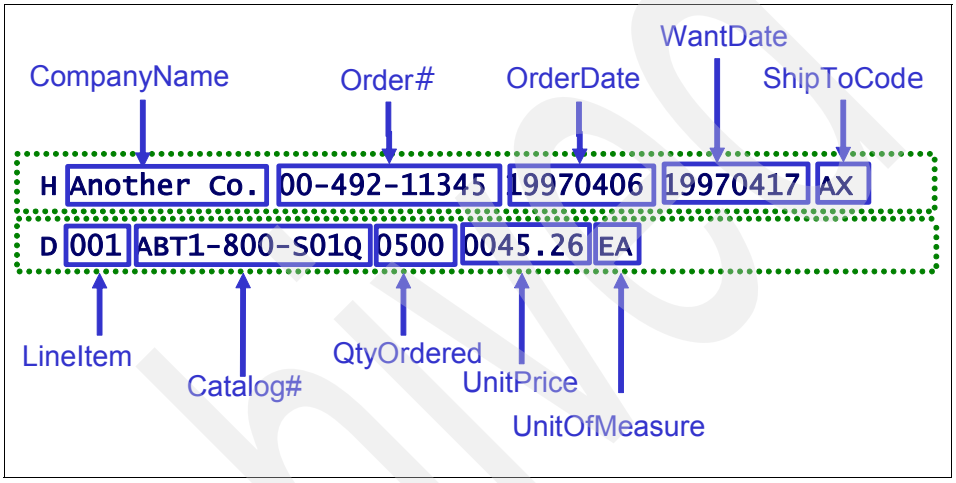

Figure 3-2 Examples of items

An object that is made up of other objects is a *group*. Groups are indicated by the icon in the type tree. Common examples of objects that are groups include records in a file, rows in a database table, a message on a queue, or an entire file or database table.

The example in Figure 3-3 on page 45 shows four examples of groups: Header Record, Detail Record, Order, and File. These objects are groups because they are made up of other objects:

- Header Record consists of a company name, an order number, an order date, a want date, and a ship-to code.
- Detail Record consists of a line item number, a unit of measure, and a unit price.
- Order consists of a Header record and a number of Detail records.
- File consists of a number of Orders.

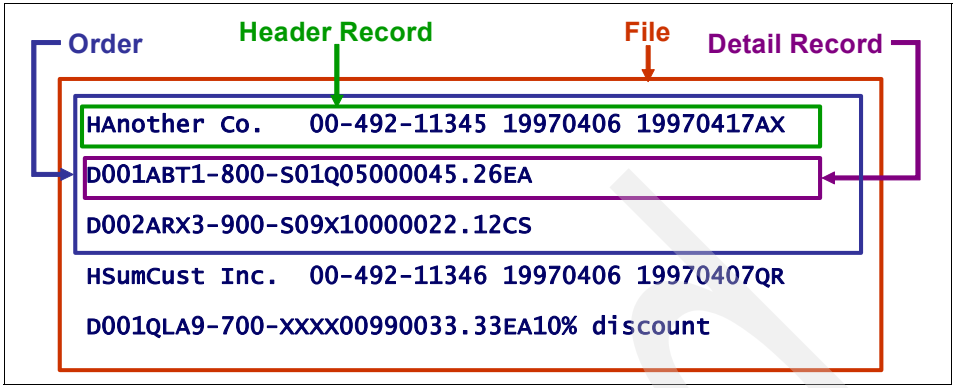

Figure 3-3 Examples of groups

Each data object is represented by a single type in a type tree. For example, in the previous example, the *CompanyName* type describes the piece of data that represents the name of the company. Similarly, the *File* type represents a number of *Orders*, or the complete data file.

While items represent simple data objects and groups represent complex data objects, *categories* are for organizational and inheritance purposes only. A category is denoted by the icon and does *not* define a specific object in data.

Relationships type

Each type branches from another type. In the type tree shown in Figure 3-4 on page 46, the *Date* type branches from the *Field* type. In this case, *Field* is the supertype (parent type) of *Date*. *Date* is a subtype of *Field*.

A subtype is a more specific definition of the type from which it branches. In this example, *Date* is a more specific kind of *Field*. *Date* might be defined as having a date/time format, while *Field* is defined simply as a text string of unlimited length. Going one level further, *Order* and *Want* are subtypes of *Date*. *Order* and *Want* are specific definitions of the more general *Date*.

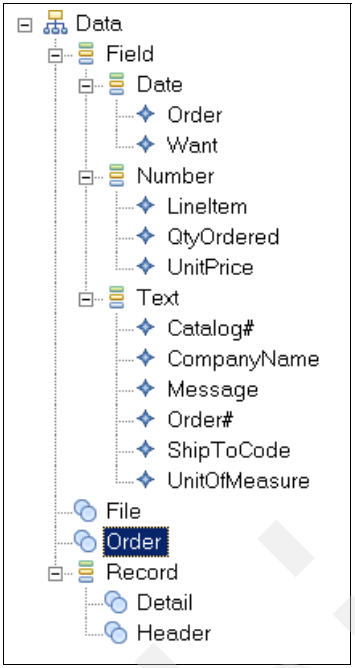

Figure 3-4 Type tree example

Subtypes are created for several reasons:

- Specific types of an object have different properties. For example, you can have two date items: one of with format MM/DD/YY and another with the format YYMMDD. These two date items can be a subtype of a more general date object.
- You might have several types of an object in the same object and want to refer to each of these types by a different name. For example, you might have two distinct dates in the same record. By creating two subtypes of date, you know by their names exactly what information they represent.

A category can have subtypes that are items, groups, or other categories. A group can only have subtypes that are other groups. An item can only have subtypes that are other items.

Assigning a class

Each data object is assigned to a class: item, group, or category. For the item and group classes, there are also subclasses. After assigning a class to a data object, you must define its appropriate format and properties. The properties are context sensitive, depending on the class and subclass selected.

For instance, if a type is an item, you must define its format, length, and so on. Each item might also have a restriction list, an initiator, a terminator, and a release character. In addition, depending on the subclass of the item (text, number, date and time, or syntax), you must define more specific characteristics.

In the upcoming sections, we discuss the class-specific formats and properties in more detail.

Inheritance: When you create a type, it becomes a subtype of whatever type you selected at the time you created the new type. Everything that defines a type (properties, components, restrictions, and so on) is passed down to the new type. This is called *inheritance*.

You can use inheritance to speed up the creation of your type tree. In doing so, keep in mind the following points:

- New types inherit properties, components, and restrictions from their supertypes.
- Categories have both item and group properties and pass appropriate properties to subtypes.

Defining properties

Assigning each type to the appropriate class is only the first step. After that, you must define the type's properties. For example, if a type is an item, you must define its subclass, size, pad, and so on. The following sections explain the type-specific properties in more detail.

Figure 3-5 on page 48 illustrates the three classes of objects that can be presented in a type tree: item, group, or category. It also shows the formats and syntax properties for groups and the interpretations and presentations for items.

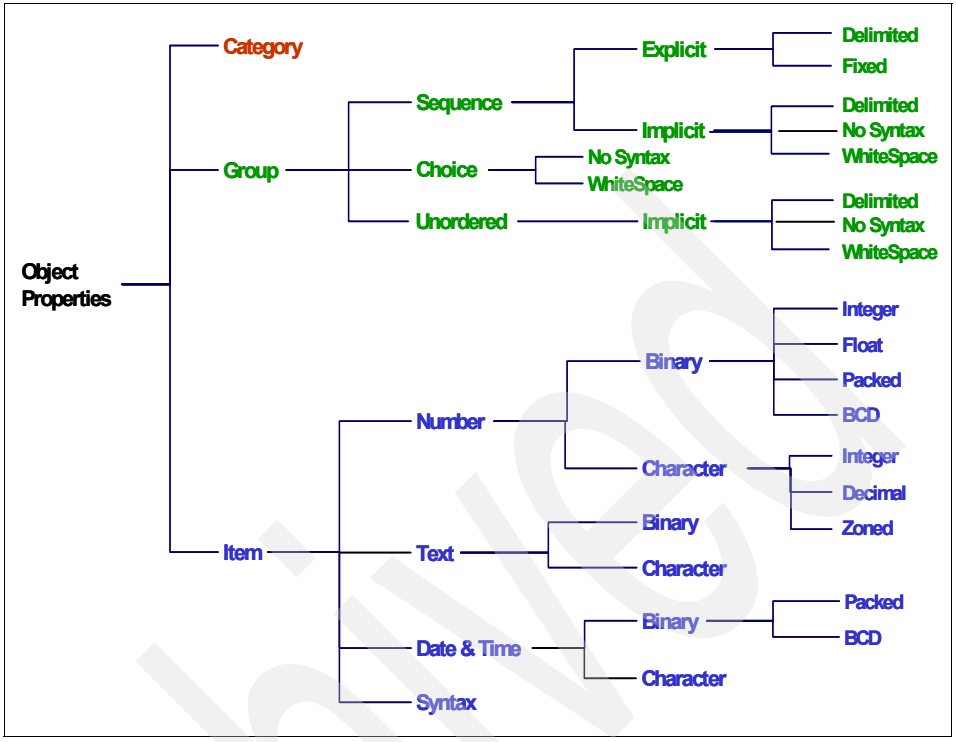

Figure 3-5 Object properties

Item class

Four subclasses can be selected to define an item's characteristics and behavior:

- Number
- Text
- Date & Time
- Syntax

Number, Text, and Date & Time are self-explanatory representations of data objects. Syntax objects are used as separators between portions of data. They represent a restricted set of values that define a dynamic delimiter, initiator, terminator, or release character. Syntax items can be interpreted as character only and cannot exceed 500 bytes. Syntax objects cannot be mapped.

Figure 3-6 shows the formats and syntax for items.

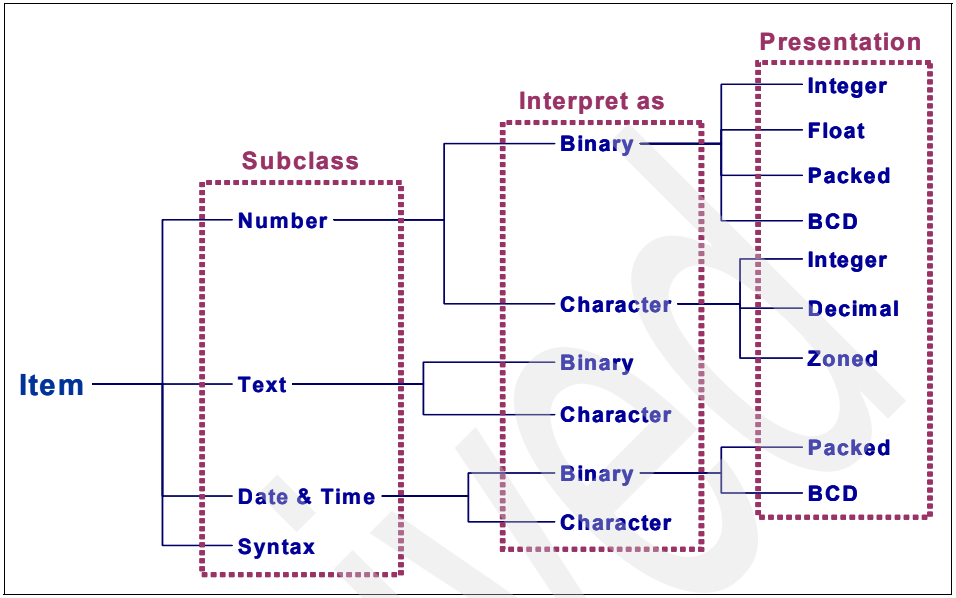

Figure 3-6 Formats and syntax for items

The properties are specific to the selected item subclass. You configure these properties in the Properties view of Design Studio. Figure 3-7 on page 52 shows the Properties view. For all items, you must define a name and a class. You should also add a description. In addition, in the Properties view for an item, the properties are organized into two large groups:

- Item Subclass, which contains the properties that are specific to the subclass selected
- Type Syntax, which is the same for all items and contains definitions for the initiator, terminator, release, and empty properties

  See “Syntax properties” on page 63 for details about these properties.

In addition, the Properties view has two more fields:

- Order subtypes, where you can select the order in which the subtypes will be presented in the type tree

  The default is ascending alphabetic order, but you can change it to descending or custom.
- Where Used, which shows where the item is used in the type tree

As you can see, several properties can be adjusted in the Properties view. In the next sections, we discuss the most used ones that are not obvious and may need some explanation:

- Interpreted as
- Size
- Pad
- Restrictions
- Number formats
- Date and Time formats

Interpreted as

An item that is defined as Text, Number, or Date & Time can be interpreted as character or binary (see Figure 3-7 on page 52):

- *Character data* is presented as a series of bytes that belong to a standard character set: ASCII, EBCDIC, or UNICODE.
- *Binary data* is presented in a series of bytes where the pattern of the bits does not map to a standard character set. Numbers are the most common example of binary item interpretation, although you might see binary dates or text items. For example, binary large objects (BLOBs) are interpreted as a binary text item.

Size

The size of a *Text* item is defined in terms of content, and you can define the number of bytes or characters for minimum and maximum sizes. See Figure 3-7 on page 52. The size excludes initiator, terminator, release characters, and pad characters. If there is no maximum size, leave the Max setting blank. For example, if an item that is a required component of a group has an initiator, that initiator must be present in the data even though the item itself has no content.

The size of a *Number* item is defined in terms of a number of digits, and you can define the minimum and maximum sizes of it.

Note: A minimum size of zero does not mean that the item is optional. An item that is optional is defined in the Group definition.

Pad

The Pad property can be set to *yes* or *no* for items that are defined as Text, Number, or Date & Time (see Figure 3-7 on page 52).

If Pad is set to *yes*, the item might contain both content and pad characters. If Pad is set to *no*, then all data is assumed to be content.

If Pad is *yes*, you must define the Value property of it (contains the Pad character). You can select either of the following options:

- One byte pad character
- Non-printable characters, for example, new line <NL>, space <SP>, and null <NULL>

<NL>: <NL> is a platform-neutral specifier for “new-line”, which represents a carriage-return or line feed on a Windows platform (<CR><LF> in WebSphere Transformation Extender) and a new-line (<LF>) on UNIX, Linux, or z/OS platform. With <NL> in WebSphere Transformation Extender, you can create a map that can be built and run on any supported platform.

However, <NL> can be confusing because we often deal with data that does not necessarily use the default platform carriage return syntax. That is, data generated on one platform might be sent to WebSphere Transformation Extender on a different platform. Sometimes it is better to define the syntax that is used in your actual data rather than try to make it platform independent.

The *Padded to* property defines whether the data item is padded to a fixed size or padded to the minimum content size defined for that data item.

If you choose the *Padded to Fixed Size* property, you must define a value for *Length*. The value must be greater than or equal to the maximum size that is defined for the item. The default is the maximum size.

If you choose the *Padded to Min Content* property, the data is padded to the number of bytes that are specified in the Min Size property of the item. If *CountsTowardMinContent* is set to *yes*, then the pad character is counted as content when validating for minimum content.

Important: A pad is not data. When an item that contains pad characters is validated, the pad characters are stripped away before any other validation occurs. If there is nothing in the data stream for a given item other than pad characters, then that item evaluates to NONE.

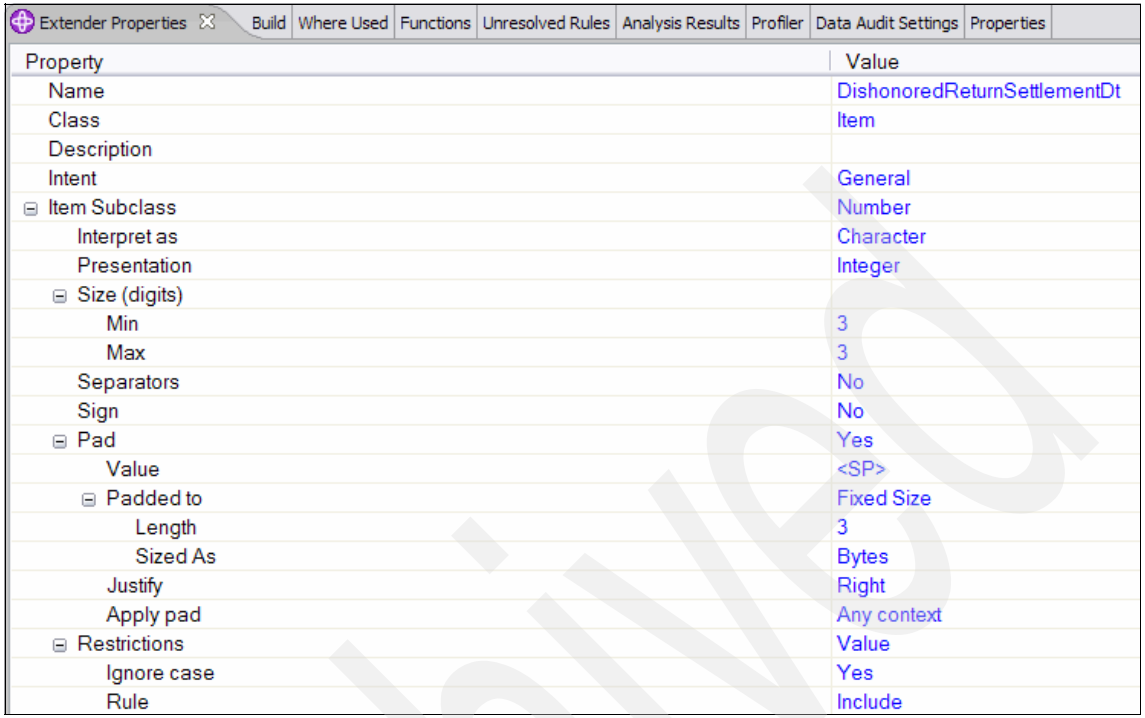

Figure 3-7 Item properties

Restrictions

When you define restrictions, you limit the valid values of that item to a specific set. You can define your restriction list to include or exclude values, characters or range. *Included values* are considered to be valid, while *excluded values* are considered to be invalid. The Restriction view varies based on restriction selections. See Figure 3-7.

Value restrictions

Consider a situation where the restriction list (Figure 3-8) of an item called UnitOfMeasure contains the values EA (each), CS (case), BX (box), and DZ (dozen).

Include	Description
EA	Each
CS	Case
BX	Box
DZ	Dozen

Figure 3-8 Restriction list

If *Restriction* is set to `Value` and *Rule* is set to `Include`, the only records that are validated are those records where UnitOfMeasure is EA, CS, BX, or DZ. If *Rule* is set to `Exclude`, the only records that are validated are those records where UnitOfMeasure is *not* EA, CS, BX, or DZ.

Character restrictions

When character text has character restrictions, and the rule is *Include*, the item window has two columns:

- One for the character list of the first character (Include First)
- One for the character list of the characters that might follow (Include After)

Therefore, the item must start with one of the values that is defined as "Include First." This value must be followed by the values that are defined in "Include After." For example, you can specify that an item must have the first character as a letter in uppercase (<LETTERUPPER/>) and the following characters must be a letter or a digit (<LETTER/> and <DIGIT/>) as in the example in Figure 3-9.

Include First	Include After
<LETTERUPPER/>	<LETTER/><DIGIT/>

Figure 3-9 Character restriction with the Include rule

An input item is considered invalid if a character other than those in the Include list is displayed in the data.

On output, WebSphere Transformation Extender builds the item with the content specified by the map rule. You might get a "`One or more outputs is invalid`" message if the item contains invalid characters.

When character text has character restrictions, and the rule is *Exclude*, the item window has two columns:

- One for the character list
- One for the reference string

The *Exclude* column contains the list of characters to be excluded. The *Reference String* is a character string that replaces the excluded character substring on output. On input, any data that contains any substring in the Exclude list is marked as invalid.

As an example, suppose that you want to output XML item content. XML character text items cannot contain the markup delimiters "<," ">," or "&" because these are reserved characters. Therefore, it must use the associated Reference String to replace these characters.

The restriction list in this example has the values as shown in Table 3-1 and is implemented as shown in Table 3-10.

Table 3-1 Example of a restriction list for XML reserved characters

Character	Replace by
<	<
>	>
&	&

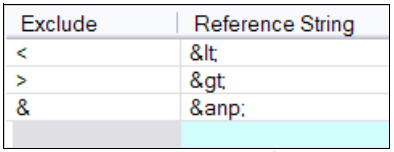

Figure 3-10 Character restriction with the Exclude rule

Range restrictions

When character text has range restrictions, the rule can be *Include* or *Exclude*. In both cases, you specify the minimum and maximum values. When the Rule property is set to *Include*, the minimum and maximum values specify the valid ranges. When the Rule property is set to *Exclude*, the minimum and maximum values specify the invalid ranges. You can also decide if the minimum and maximum values that you have specified are in or out of the interval. (By default, the value is considered included.) See Figure 3-11 for an example.

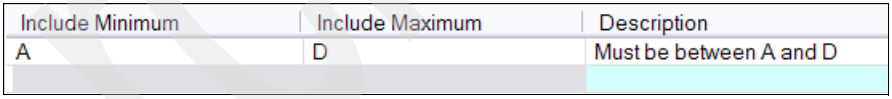

Figure 3-11 Range restriction with the Include rule

Range restrictions are valid for numbers and dates.

Number formats

A number is defined in terms of a minimum and maximum number of digits. The size includes a sign or separators. If there is no maximum size, leave Max setting blank.

When a number presentation is defined as a decimal, you also have the option to define a format, specifying the number of decimal places if the number contains “.” and “,” separators for decimal and thousands, and so on.

Note: When the content size of a character number is important, such as when using the SIZE function, the value returned includes separators and a sign.

Date and Time formats

When the item subclass is set to Date & Time, the options to specify a minimum and maximum size are unavailable. They are unavailable because the selected date format determines the size of the data object. For example, if a Data & Time format of CCYYMMDD is selected, the item size is implied to be eight characters.

In addition to the predefined format, you can specify a custom Date & Time format. After you define and save a custom format, the custom format string is displayed in the Properties view.

Table 3-2 shows the Date component options that are available.

Table 3-2 Date component options

Symbol	Description
CCYY	Full year, including century
YY	Last two digits of the year (00-99)
MM	Month of year in two numeric digits (01-12)
M	Month of year in one or two numeric digits (1 to 12)
MON	First three letters of month (Jan to Dec)
MONTH	Full name of month (January to December)
DDD	Day of year in three numeric digits (001 to 366)
DD	Day of month in two numeric digits (01 to 31)
D	Day of month in one or two numeric digits (1 to 31)
DY	First three letters of weekday (Sun to Sat)
DAY	Full name of weekday (Sunday to Saturday)
WW	Week of year (1-52)
Qn	Quarter of year (Q1-Q4)
EEYY	Emperor's year long form
EY	Emperor's year short form

Table 3-3 shows the available Time component options.

Table 3-3 Time component options

Symbol	Description
HH12	Hour (based on a 12-hour clock) in two digits (1-12)
H12	Hour (based on a 12-hour clock) in one or two digits (1 to 12)
HH24	Hour (based on a 24-hour clock) in two digits (00 to 23)
H24	Hour (based on a 24-hour clock) in one or two digits (0 to 23)
MM	Minute in two digits (00 to 59)
M	Minute in one or two digits (0 to 59)
SS	Seconds in two digits (00 to 59)
S	Second in one or two digits (0 to 59
AM/PM	Meridian (AM/PM)
+/-ZZZZ	Hours and minutes before or after Greenwich mean time (GMT)
ZZZ	Three character abbreviation for time zone (EST, and so on)

Group class

Figure 3-12 shows the formats and syntax for groups.

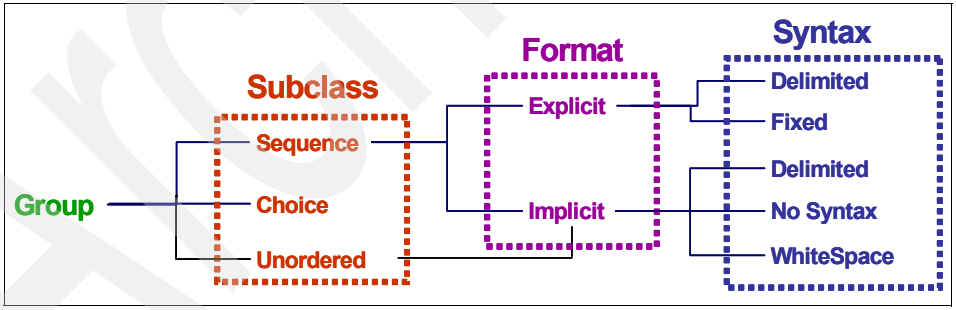

Figure 3-12 Group subclass formats

There are three kinds of groups:

- *Sequence groups* are a partially-ordered or sequenced group of data objects that are denoted by the icon.
- *Choice groups* provide the ability to define a selection from a set of components and are denoted by the icon.

- *Unordered groups* allow components to be displayed in any order and are denoted by the  icon.

Sequence group formats

Group properties include the group's format, which describes how to distinguish one component of that group from another component of that group. A group has either an explicit or implicit format.

The format of a group is *explicit* if the components are distinguishable because of a delimiter or their position. A group has an *implicit* format if its components are distinguishable because of their own definitions.

An *explicit group* can have either a delimited or fixed syntax.

An explicit group has *delimited syntax* when the components are separated by a delimiter. A *delimiter* is a character (or series of characters up to 500 bytes) that separates the data objects. The delimiter is displayed as a placeholder even when the component has no content.

The only time a delimiter can be missing is if the components that follow the delimiter are all optional and there is no data for these optional components.

A delimiter can be defined as either a literal or a variable:

- Literal, if the delimiter is a constant known value

  The value of the actual delimiter is defined in the Value property.

- Variable, if the delimiter value can vary

  You can define this delimiter as an item in the type tree and list all the possible values.

Delimiters can be in one of three locations with respect to the components of the group:

- A prefix delimiter is displayed *before* each component (^A^B^C).
- A postfix delimiter is displayed *after* each component (A^B^C^).
- An infix delimiter is displayed *between* each component (A^B^C).

An explicit group has a fixed syntax when the group object is always the same size (in bytes). In order for a group object to be of fixed length, *all* of its components must be of a fixed length. Ultimately, a fixed group is made up of items that are padded to a fixed size. One component is distinguished from the next by the fixed size of each component.

Component range: A component range indicates the number of occurrences of a component. The component range is displayed after the component name in parenthesis and is specified as two numbers separated by a colon (Min:Max). The first number is the minimum number of consecutive occurrences of that component, and the second number is the maximum number of consecutive occurrences of that component. To specify an indefinite number of occurrences, specify “s” as (s), which means “Some.” See “Component range” on page 67 for full details.

Guidelines for defining a fixed group: Follow these guidelines to define a fixed group:

- Each component must be a group with a fixed syntax or a fixed item, where the item is padded to a fixed size or its minimum and maximum content size are equal.
- Each component must have a specified range maximum, and the maximum cannot be “s.”
- If a component range minimum is not equal to the maximum, content is not required for optional component occurrences.

For example, if a component is the Item ShippingAddress (0:1) and ShippingAddress has a minimum content size of two characters, data can contain all pad characters. Or if content is in the data stream, the data must have a minimum of two characters for a ShippingAddress. If a component is a group, no content is required for any of the items that are in an optional occurrence of that component.

An implicit group can have no syntax (None), delimited syntax, or whitespace syntax.

In a group with an implicit format *syntax of None*, the components are distinguishable not by delimiter or position, but by their pattern, which is something in the definition of the component types themselves. The group has no syntax property that distinguishes one component from another. In this case, the following points are true:

- Components are not separated by a delimiter.
- Group object is not a fixed size.
- Component properties distinguish one component from another.

For example, take a situation where the file type is made up of records. An <NL> is displayed at the end of each record, which has been defined as the terminator

of the record. However, the file has no syntax of its own. The map relies on finding the record terminator to distinguish one record from another.

Figure 3-13 shows an example of a group with an implicit format and a component syntax of None.

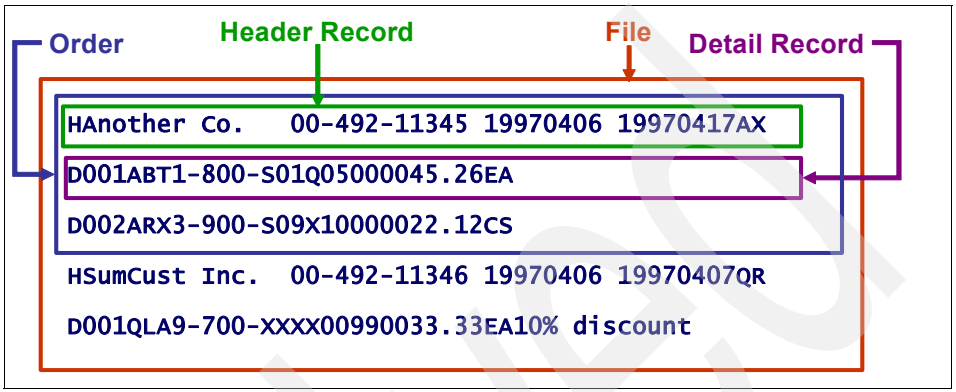

Figure 3-13 Group with an implicit format and a component syntax of None

In this example, note the following points:

- The file is made up of orders.
- One order is distinguishable from the next because each order starts with a header record (records starting with an “H”), followed by some variable number of detail records (records starting with a “D”).
- The components of the file distinguish one from another by the pattern of the records that make up each order.

Rule of thumb: When deciding what format a group has, use this rule of thumb: If the group is not explicitly fixed or delimited, it is implicit.

If an implicit group has a *delimited syntax*, define the Delimiter properties as described for explicit groups with a delimited syntax. In an implicit delimited group the delimiter is not displayed as a placeholder when the component has no content.

Implicit delimited groups can be useful for certain kinds of data, such as tagged data. In the example shown in Figure 3-14 on page 60, Employee can be described as an implicit delimited group, where the delimiter is <NL> and the components are FirstName, LastName, and Title. Notice that, in the second instance of the Employee, Title is missing, but there is no placeholding <NL> in the data.

```
<Employee>
<FirstName>Jane</FirstName>
<LastName>Doe</LastName>
<Title>Technical Sales</Title>
</Employee>
<Employee>
<FirstName>Chang</FirstName>
<LastName>Wu</LastName>
</Employee>
```

Figure 3-14 Group with an implicit format and a component syntax of delimited

If an implicit group has a white-space syntax, white spaces are not allowed in the data. If you select the WhiteSpace option, define the Build As property to define which characters will replace the white spaces.

The flow chart in Figure 3-15 can help you to decide which group format and syntax to use. The NO* just before “Implicit, Delimited” in the figure means that, in some cases, such as when optional components are missing at the end of a record and the ending delimiters are omitted, the group format and syntax are “Implicit, Delimited.”

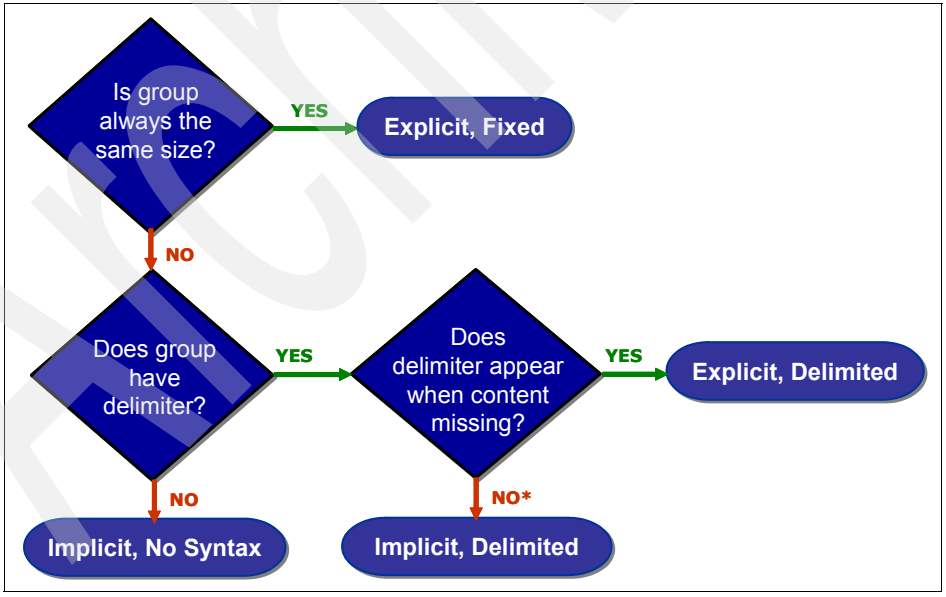

Figure 3-15 Group format and syntax options

Choice group format

Choice groups provide the ability to define a selection from a set of components similar to a multiple-choice question on a test. Choice groups are similar to partitioned sequence groups. However, a choice group can have both items and groups as components. A partitioned sequence group can only have group subtypes. For more information about partitions, see “Partitioning” on page 62.

A choice group data type is only one of the group components. Therefore, the components of a choice group cannot have a component range other than (1:1). For example, consider a record data type as a group type with a group subclass of choice. The record group type has three components: order, invoice, and sales. The data validation of record is only one of the components: order, invoice, or sales.

On input, a choice group is validated as only one of its components. Validation of a choice group is attempted in the order of the components until a single component is validated. If the choice group has an initiator, the initiator is validated first.

On output, only one component of a choice group is built in the output data.

Choice groups have no Partition or Format properties. A choice group can have a component syntax of None or WhiteSpace. Type syntax properties, such as Initiator, Terminator, Release, and Empty, can be applied to choice groups. If a release character is defined for a choice group, by default, it applies to the Terminator property.

If the group has a white-space syntax, white spaces are not allowed in the data. If you select the WhiteSpace option, define the Build As property to define which characters will replace the white spaces.

Unordered group format

An unordered group has one or more components that can be displayed in the data stream in any order. It has no partitioned property and is implicit in format. Syntax options for unordered groups are the same as for sequence groups.

Unordered groups allow a more natural syntax for typing and mapping data that occur in the data stream in an unpredictable manner, such as applications that use SWIFT standards or XML.

The key issue in using unordered groups successfully is the ability to ensure that components of the group can be distinguished from each other and from other components of the data stream. Remember that partitioning is a good way to create distinguishable objects, so that the components of unordered groups can

often be partitions of another partitioned group. For more information about partitions, see “Partitioning” on page 62.

One distinct advantage that unordered groups have over partitioned groups is that unordered groups allow ranges to be specified. For example, the unordered group, A, has the following component list:

```
x, that occurs once only.
y, that occurs one or more times (1:s).
z, that occurs zero or more times (s).
```

In this case, the data stream must have one x, at least one y, and possibly more than one z. For example, the data stream can be any of the following combinations:

```
xy
yzzyxzyz
zyyyzyx
```

Unordered groups have no Partition property. They have Implicit format properties with the same syntax options as sequence groups: None and Delimited. Unordered groups can also have syntax properties: Initiator, Terminator, Delimiter, and Release character.

Upon input, the components in an unordered group are validated according to the type and range. Upon output, an unordered group is built according to the following rules:

- Each component is built as though it were ordered, and the order is based on its position in the component list.
- If an unordered group component occurrence is required and it has a rule that evaluates to NONE, its type syntax objects are built.
- If an unordered group component is optional and has a rule that evaluates to NONE, its type is not built.

Component rules of an unordered group cannot reference other components of the same group. They can only reference the component to which the rule refers and the objects contained in that component.

Partitioning

When a data object can belong to any number of different types, there must be a way to identify the difference between them. *Partitioning* is the method of subdividing groups into mutually exclusive subtypes to improve processing and the validation of data. An example of partitioning is to manage data that can arrive in random order, such as a collection of unrelated records or attribute/value pairs within an XML element.

The example data file in Example 3-1 represents unordered data. The file contains three different types of transactions: invoices, orders, and remittances. Each of these transactions has a different definition based on the type of information that it represents.

Example 3-1 Unordered data file

```
BGI-13100,REM,931104,19970424…
AXR-10930,INV,003X-114,19970422…
PVY-19496,ORD,PO-104-1499,19970425…
BGI-13100,ORD,PO-182-2587,19970425…
AXR-10930,INV,003X-114,19970422…
PVY-19496,REM,931104,19970424…M
```

In this example, the second field in each record identifies the type of data that is contained in the record:

- REM indicates that the record is for a remittance.
- INV indicates that the record is for an invoice.
- ORD indicates that the record is for an order.

We can partition this object into the three different transaction types. As each transaction is validated, the Transformation Extender engine determines the partition to which it belongs and validates it according to that partition's definition.

Partitioned groups versus unordered groups: Partitioning is a property that applies only to sequence groups. A partitioned group subdivides objects into mutually exclusive subtypes.

Unordered groups address identification of components that can be displayed in any order in their data stream and are not mutually exclusive. In an unordered group the values occur in any order within the data stream, but the objects must be distinguishable and are not mutually exclusive.

Syntax properties

In addition to the Item and Group subclass properties, you must define the Type Syntax properties. The Item Subclass properties are specific for the subclass of the Item (Text, Number, Date & Time, or Syntax), and the Group Subclass properties are specific for the subclass of the Group (Sequence, Choice, or Unordered). However, *all* the types in the type tree have the same Type Syntax properties: Initiator, Terminator, Release, and Empty.

Initiator

The Initiator property represents the syntax that is displayed at the beginning of a data object. It is defined in type tree properties under **Type Syntax** → **Initiator**.

When you define an initiator, you indicate that whenever you see data of that type, this initiator is displayed at the beginning of it. An initiator can be defined as a literal or as a variable.

The example in Figure 3-16 shows two types of records: Header records and Detail records. Header records have an initiator of H. Similarly, D is defined as the initiator for the Detail record type.

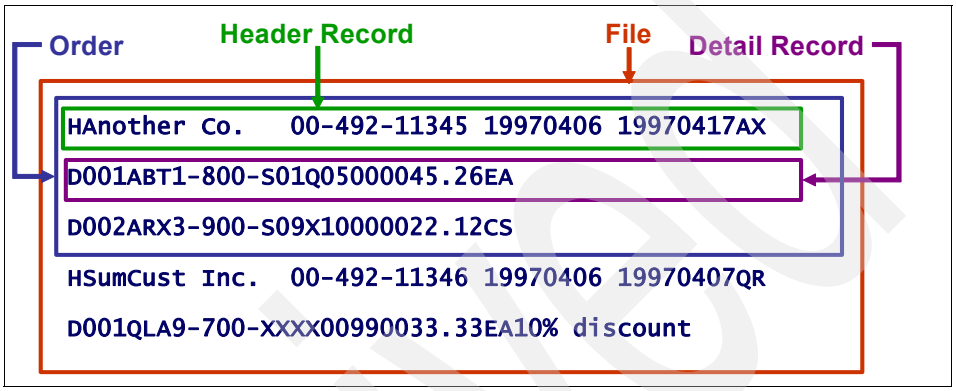

Figure 3-16 Example of the initiator type syntax

Terminator

The Terminator property represents the syntax that is displayed at the end of a data object. It is defined in type tree properties under **Type Syntax** → **Terminator**.

When you define a terminator, you indicate that whenever data of that type is displayed, this terminator is displayed at the end of it. A terminator can be defined as a literal or as a variable.

For example, in a file made of order records, if each record has a new line character <NL> at the end, then you must define this character as the terminator of the group object.

Release

A *release character* is a character in the data that indicates that the characters that follow it should be interpreted as data, not as a syntax object. A release character can apply to a delimiter, a terminator, and the release character itself. A release character can be defined as a literal or as a variable. It is defined in the type tree properties under **Type Syntax** → **Release Character**.

In the example in Figure 3-17, a group is defined as having a delimiter of comma (,) and a release character of ampersand (&). The & indicates that the , that follows it is data and *not* a delimiter.

```
Release character = Ampersand (&)
Delimiter = Comma (,)

Data = Doe&, Jane,IBM,Technical Sales
```

Figure 3-17 Example of release character

If this data is displayed in an input data stream, it is interpreted as follows:

```
Data of component 1 = Doe, Jane
Data of component 2 = IBM
Data of component 3 = Technical Sales
```

Suppose that the release character of & was not defined in the Record group. The data shown in the input data stream is interpreted as follows:

```
Component 1:Doe&
Component 2:<sp>Jane
Component 3:IBM
```

The remaining data (",Technical Sales") is unknown and might result in the entire record being seen as invalid.

Considerations regarding release characters:

- Release characters apply to character data, not binary data.
- A release character releases the characters that make up the entire delimiter or terminator even if the delimiter or terminator is more than one character.
- Release characters that are not followed by an associated active syntax object are ignored.
- Characters that are used as pad characters are not released.

Empty

The Empty property provides alternative type syntax for groups or items when they have no data content. It is defined in type tree properties under **Type Syntax** → **Empty**.

When the Empty property is specified for a type and there is no data content, the Empty syntax is displayed. For example, this property can be used for XML data that contains either start and end tags or an empty tag.

The following options are available for the Empty property:

None The default setting. Select None if you do not have an initiator, terminator, or release character.

Literal A constant value. When Literal is selected, the Value, Ignore Case, Required, and National language subfields become available. You can use only one literal to indicate a zero-length data item.

Defining components

The type tree view shows only a list of the types that make up the data and their hierarchy. It does not show anything about the composition of the data. For example, you cannot tell what the layout of an order is, nor can you tell what makes up a file. Therefore, after defining the properties of all the elements of a type tree, you must define the components of the group elements in the type tree.

A component is a data object that is part of a complex data object. In terms of a data type, a component can be an item or a group. Each group *must* have at least one component. Items cannot have components because it is the simplest data object. Remember that items are data types that are *not* made up of other data types. Categories do not need components. A category organizes types or provides for inheritance.

To determine the components of a group, ask yourself: What makes up this complex data object? By listing the components of an object, you are defining its structure. The order of components in the definition of the complex data object must reflect the order of the component objects in the data stream.

A group and its components *must always* be in the same type. Figure 3-18 on page 67 shows the components of a group that represent a file. You can see that each of the components can be further composed of other components. You can also see that the components have a range, and some can have a rule assigned to them. The component range and the component rules are described in the sections that follow.

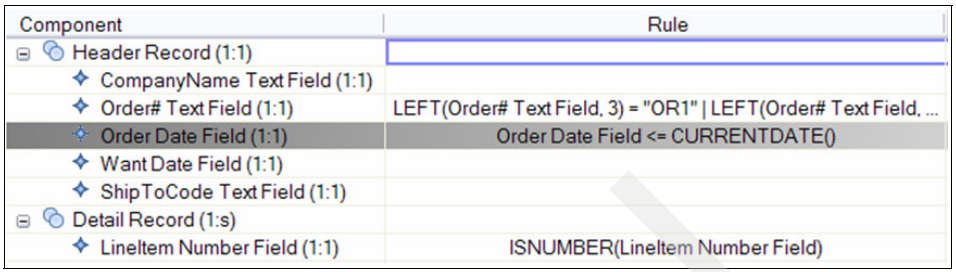

Figure 3-18 Components of a group

Component range

Each component in the component list must be interpreted as a unique object. Therefore, each component must have a unique name. The range is used to indicate the number of occurrences of a component.

For example, suppose that we want a group to contain two messages. Since it is impossible to have a duplicate name, the range is used to indicate that the field can be displayed more than once consecutively. In this case, the range for the message is (1:2). If the messages are considerably different, it is better to create two different message types.

The *component range* specifies the number of consecutive occurrences of that component that can be displayed at that point in the data stream. The component range is displayed after the component name in parenthesis and is specified as two numbers separated by a colon (Min:Max):

- The first number is the minimum number of consecutive occurrences of that component. This value is optional. If it is not specified, it is interpreted as a minimum of 0.
- The second number is the maximum number of consecutive occurrences of that component.
- To specify an indefinite number of occurrences of a component, specify the maximum as “s,” which means “Some.” For an example, see Detail Record in Figure 3-18.

Component rules

A *component rule* is an expression or statement about one or more components in a component list. The component rule indicates what must be true for that component to be valid. Figure 3-18 shows examples of component rules in the right column.

In a component rule, you can reference the component itself or any other component earlier in the same component list. This referencing is possible

because the WebSphere Transformation Extender engine parses the data based on the order of the components.

Any reference to a component in its component rule (the rule next to that component in a component list) can be replaced by the dollar sign ($). Component rules are always evaluated, even if an optional component is not present in data. If the component rule is only evaluated when data is present, then use the WHEN(PRESENT…) function.

Output objects are only validated when data audits are defined for that output and Data Log is enabled.

Analyzing type trees

After you finish creating the type tree that defines your data, you use the Type Tree Analyzer in Design Studio to ensure that the definitions in the type tree are internally consistent.

You can choose to analyze the structure or the logic of your type definitions, or both:

- Logical analysis

  Logical analysis addresses the integrity of the relationships that you define. For example, it detects undefined components, components that are not distinguishable from one another, item restrictions that do not match the properties of that item, and circular type definitions. The analyzer also checks delimiter relationships to each other and to components, undefined inherited relationships, and logic errors in the component rules.

- Structural analysis

  Structural analysis addresses the integrity of the underlying tree structure. Generally, you should not encounter structural analysis errors. Structural analysis might be able to detect and possibly correct defects that are caused by system environment failures.

The Type Tree Analyzer does *not* compare definitions to actual data. This comparison is done in a validation map.

All type trees must be analyzed before they can be used in a map. All errors must be resolved, while the warnings indicate an inconsistency that *should* be resolved.

3.2.2 Importers

The Importer Wizard can import various data formats to create type trees (which have the .mtt extension). Many of the type trees that are generated by the Importer Wizard can be immediately used for map development. However, depending on the contents of the interface-specific metadata file, it might be necessary for the generated type tree to be modified by using Design Studio.

The following importers automatically create the data definitions from several external metadata sources:

- COBOL Copybooks Importer
- CORBA IDL Importer
- Java Class Importer
- Java Messaging Service (JMS) Importer
- PL/I Include Importer
- Text File Importer
- Type Library Importer
- Web Services Description Language (WSDL) Importer
- XML DTD Importer
- XML Schema Importer

In addition, the following importers are part of the WebSphere Transformation Extender Enterprise Packages:

- For SAP, there is BAPI, Idoc, DXOB, and BDC.
- For PeopleSoft 8, there is Component Interface.
- For Siebel, there is Siebel Business Object.

COBOL Copybook Importer

The COBOL Copybook Importer imports the definition of one or more COBOL copybooks in a file and generates a type tree that contains the corresponding type definitions. The copybook file can contain more than one COBOL copybook. If your file contains multiple COBOL copybook definitions, all definitions are in a single type tree.

CORBA IDL Importer

The CORBA Interface Definition Language (IDL) Importer is used to generate a type tree representation of an IDL file.

Java Class Importer

The Java Class Importer creates a type tree that can be used to manipulate Java objects from within a map by using the Java Class Adapter.

JMS Importer

The JMS Importer creates type trees that represent the public fields and constructors of Java classes that are sent as objects by using JMS messages.

PL/I Include Importer

The PL/I Include Importer imports the definition of a PL/I include file and generates a type tree that contains the corresponding type definitions.

Text File Importer

The Text File Importer automatically generates type trees that describe the format of the data in text files. The Text File Importer executes a wizard, with which you can graphically mark a text file to select and create the components of the type tree. For example, you can select and create each group and item and any delimiters or terminators. You can set all of the properties for each element.

Type Library Importer

The Type Library Importer reads the format of COM Components and Methods to automatically generate type trees for use with the COM Automation Adapter.

WSDL Importer

The WSDL Importer is a utility for generating type trees from message definitions that are in the WSDL documents. The output that is generated by the WSDL importer consists of a type tree and an XML map source file.

XML DTD Importer

Use the XML DTD Importer to create type trees that describe the format of XML data based on an existing DTD.

XML Schema Importer

Use the XML Schema Importer wizard to generate a type tree from an XML schema.

Note: The XML Schema Importer only supports schemas that conform to the 2001 XML Schema specification that is defined by the Worldwide Web Consortium (W3C). Other schema options, such as BizTalk® or XDR, are not supported. However, some providers, such as BizTalk, provide tools that can convert a proprietary schema to the latest W3C XML schema specification. See the latest WebSphere Transformation Extender product documentation to review any limitations on the XML Schema Importer at the following address:

`http://www-01.ibm.com/software/integration/wdatastagetx/library/index.html`

Importers with a JNI™: Importers with a Java Native Interface (JNI) use the Java virtual machine (JVM™) options specified in the `dtx.ini` file to create a JVM. You can use the JVM options to tune the importer's behavior, such as setting up a big initial or maximum heap size.

The following importers have JNIs:

- CORBA IDL Importer
- Java Class Importer
- JMS Importer
- WSDL Importer
- XML DTD Importer
- XML Schema Importer

The following JVM options can be modified in the `dtx.ini` file to fit your specifications:

- JVM options
- JNI layer trace

Another component that can be used to automatically generate type trees is the Database Interface Designer. It is used to import metadata about queries, tables, and stored procedures for data that is stored in relational databases. It is also used to identify characteristics of those objects to meet mapping and execution requirements such as update keys and database triggers. See 5.3.8, "Database Interface Designer" on page 296, for details about Database Interface Designer.

Despite the way you create your type tree, all definitions are modifiable through Design Studio.

3.2.3 Type Tree Maker

Another way to create type trees is to use the Type Tree Maker. The Type Tree Maker is an application that is installed with Design Studio. It reads in an XML document file (.mts extension) that contains type tree commands and uses these commands to create or update a type tree.

The script file (.mts extension) that is used by the Type Tree Maker to create the type tree contains commands and command elements that must be specified in a particular way and in a specific sequence. Check the documentation in the online library at the following address for details about Type Tree Maker commands and command elements that make up the Type Tree Maker document (script) file:

`http://www-01.ibm.com/software/integration/wdatastagetx/library/index.html`

3.3 Maps

An executable map contains the complete definition of data structures. The data structures represent the source and target, the rules used for transformation, and specifications of resource adapters that are used to retrieve and send data.

3.3.1 Map files

To develop a map, we use the Map Editor, which is a part of WebSphere Transformation Extender Design Studio. A map source file has the .mms extension and can contain any number of maps. Before you can use a map, you must build a map. The build process compiles and optimizes a map for execution on a specific platform. A map source file can contain many maps and is platform independent. However, the compiled map file contains only one executable map and is designated for one specific platform. If you want to use the same map on different platforms, such as Windows and z/OS, you must compile it for each platform independently.

Figure 3-19 shows an example of map files.

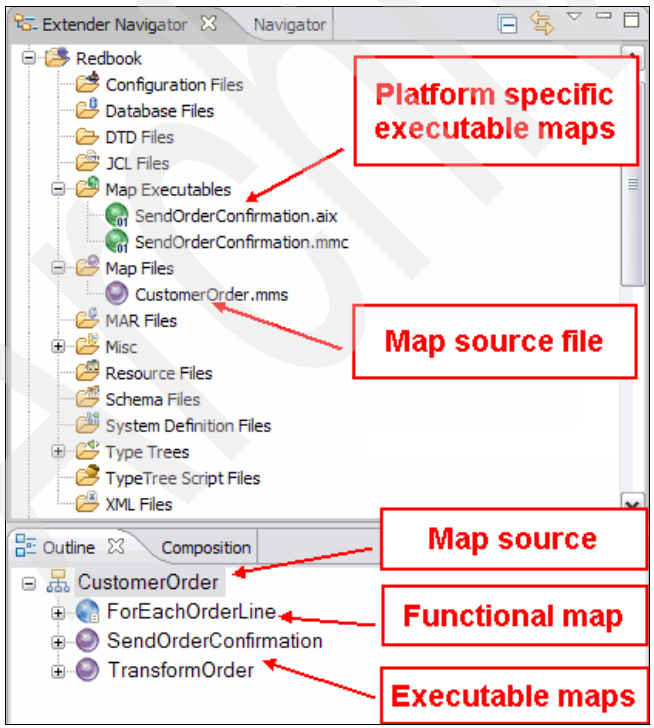

Figure 3-19 The Extender Navigator view in Design Studio

As shown in Figure 3-19 on page 72 in the Outline view, the `CustomerOrder.mms` map source file contains three maps: ForEachOrderLine, SendOrderConfirmation and TransformOrder. Notice the different icons between the maps. The ForEachOrderLine map is a functional map, and the two other maps are executable maps. We explain the difference between functional and executable maps in 3.3.4, “Functional maps” on page 82.

Also shown in Figure 3-19 on page 72 is the SendOrderConfirmation map, which has been compiled twice for different platforms:

- For IBM AIX®, the `SenderOrderConfirmation.aix` file
- For Microsoft Windows, the `SenderOrderConfirmation.mmc` file

A map has the following components:

- Input and output map cards that represent the sources and targets
- Map rules that define how the output data is built based on business rules
- Specifications of the resource adapter used to retrieve and send data

3.3.2 Cards

Each map can contain any number of input and output cards. A single card represents a single data source or target. The structure of the data is specified in the card settings, and it corresponds to a type from the type tree file. Each input or output card should represent one data source or data target. For example, if your map takes data from two different files and a database table, you should add three input cards. An input and output card are similar, yet unique:

- An *input card* contains the complete definition of an input for the map including information such as source identification, retrieval specifics, and the behavior that should occur during processing.
- An *output card* contains the complete definition of an output for the map including information such as target identification, destination specifics, and the behavior that should occur during processing.

The complete information about each card is specified in the card settings. Each card is identified by the card name. The card name is case sensitive and must be unique for the whole map.

Figure 3-20 shows a map that contains three input cards and one output card.

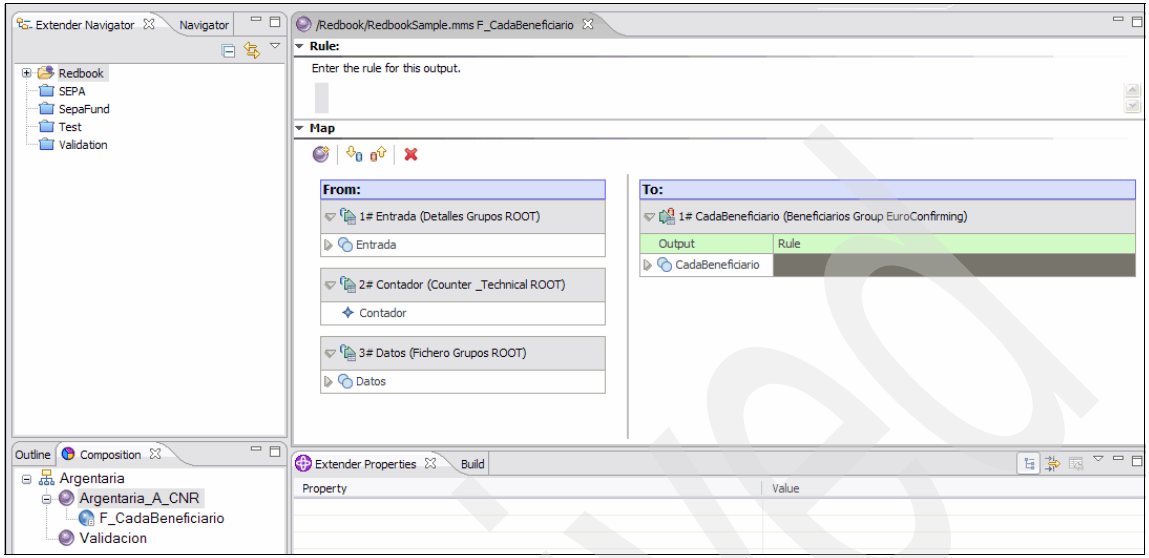

Figure 3-20 An example of map that contains three input cards and one output card

Each card in a group has a number, and the numbers are assigned sequentially when the card is created. However, later cards can be reordered.

- For output cards, the card number determines the order of rules evaluation. The order is important because, in mapping rules, you can refer to the previous cards, but not to further cards.
- For input cards, the following situations apply:
  - If the map is an executable map, the card number determines the sequence in which the input data are validated.
  - If the map is a functional map, then the order of input cards is important, because the sequence of input arguments depends on it.

Card settings

The card settings in the Schema section, in addition to the CardName parameter, contain the Type Tree and Type parameters. These parameters define the content of the card, but do not specify how the data is retrieved for input cards nor how it will be produced for output. The SourceRule section for input cards defines where are the data comes from and how it will be retrieved. A similar set of parameters for the output cards is available in the Target Rule section. The parameters from the FetchAs section are accessible only for input cards, because they define the behavior of the map during the fetching and validation process.

Figure 3-21 shows the card settings for input and output cards.

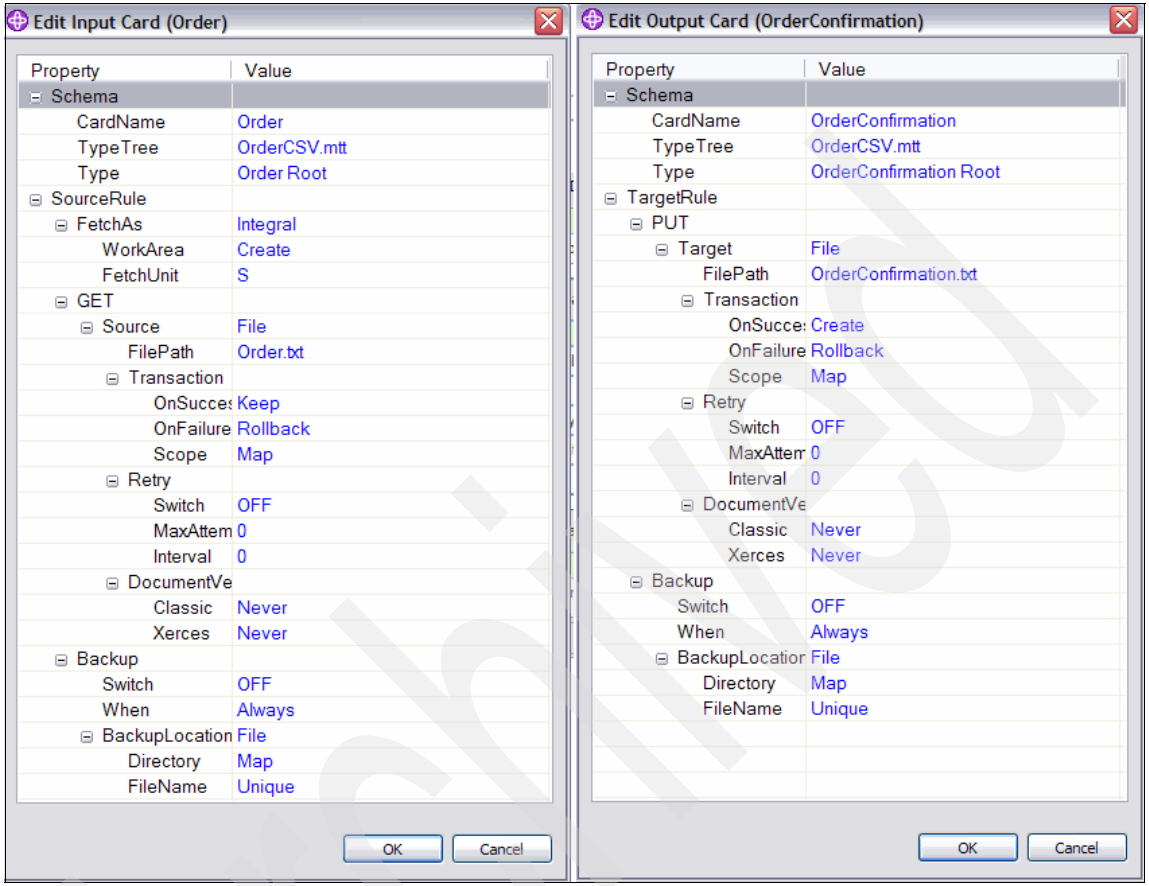

Figure 3-21 The card settings for input and output cards

In the Source and Target properties, we specify the adapter that is used to retrieve or store data. In the previous example, we use the File adapter. The list of possible adapters is long (Figure 3-22 on page 76) and contains all of the most commonly used methods of communications. Most of the parameters in the SourceRule or TargetRule can be overridden at execution time. By overriding these parameters, you can dynamically change the map behavior without modifying and recompiling maps.

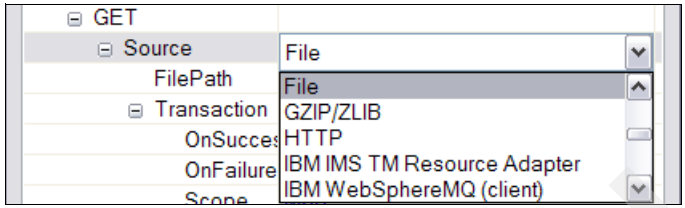

Figure 3-22 Partial list of adapters in the card settings

In case you are using XSD schema files as a type tree, or type trees that are imported from XSD with the Xerces option, the additional Metadata section is accessible in the card settings, as shown in Figure 3-23. In this section, you can define the namespace aliases, which are used in the generated XML document.

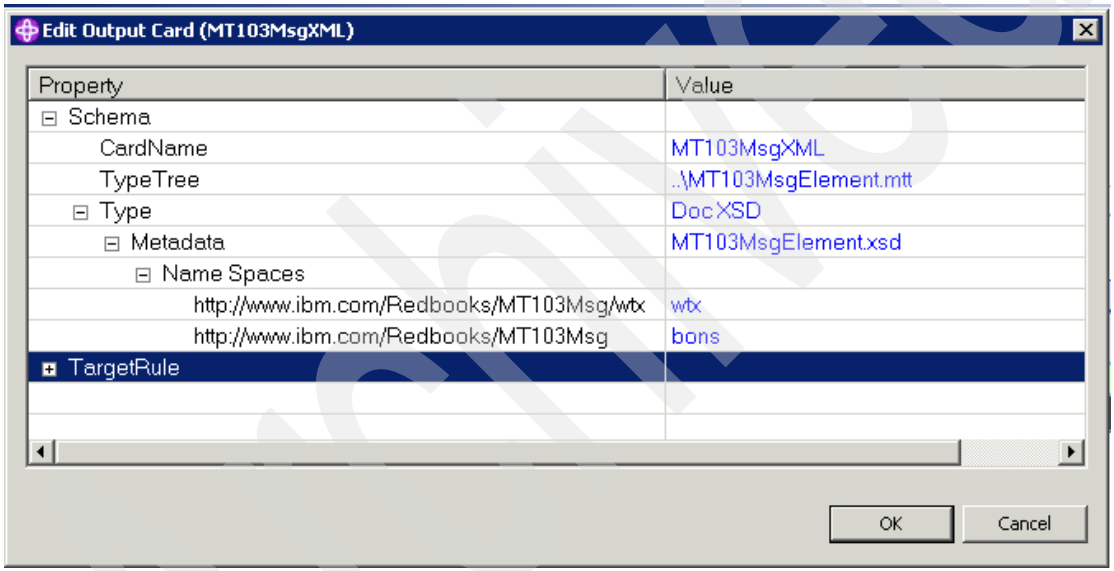

Figure 3-23 The Metadata section for XML type trees

XSD schema files: The Metadata section is visible only if the XSD schema file is in the same folder as your map source file. This section is necessary only during the development process. If you do not want to place the XSD files in the runtime environment, you must switch the *DocumentVerification* property to *Well formed (Xerces only)*, as shown in Figure 3-24 on page 77. For type trees that are imported from XSD schema files by using the *Classic* option, the schema file is needed only during importing and does not need to be kept with other project files.

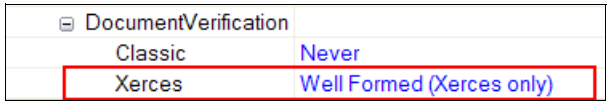

Figure 3-24 The Document Verification property

The Composition view of the cards shows the contents of the type definitions. Types are arranged in a compositional hierarchy, which shows the structure, layout, or composition of the data. This hierarchy is different from the hierarchy in a type tree, which is a classification hierarchy. The types are arranged on the input and output cards in the order in which they are displayed in the data. The compositional hierarchy in Figure 3-25 shows the structure of the data.

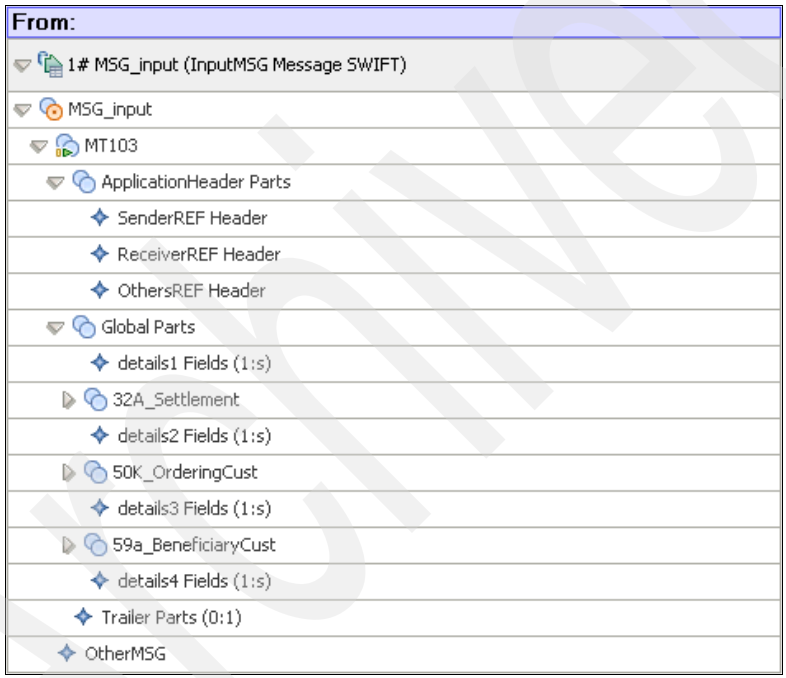

Figure 3-25 Hierarchy of the data

3.3.3 Rules

The Output card view has a Rule column for specifying the mapping rules (Figure 3-26). A *rule* is an expression that describes how the output data should be built.

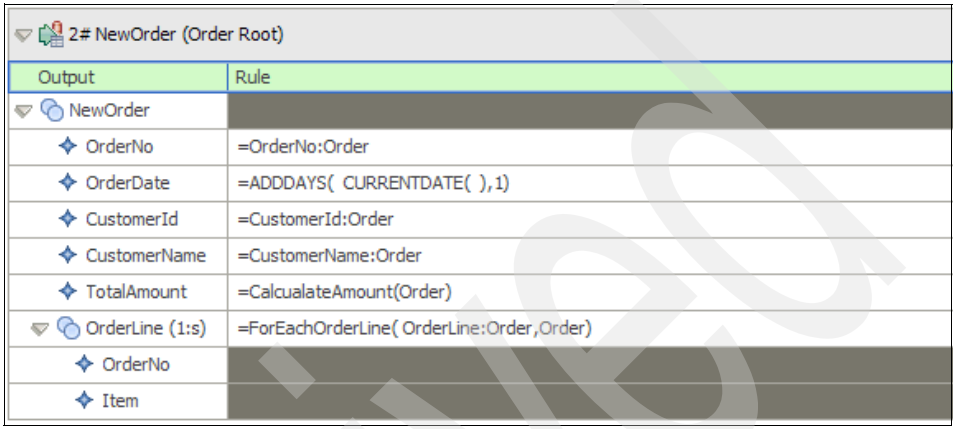

Figure 3-26 The Output card with mapping rules

All output fields should have a rule provided. If you do not need to generate an output object, a map rule of =NONE is required. Empty rule cells generate an error during the map building.

You can build your output rule for several purposes as in the following examples for a map rule:

- Map an input object to an output object.
- Extract input records based on specific criteria.
- Count a conditional number of records.
- Map input records based on a specific qualifier to a specific output record.
- Convert an object in input data from one value to another value.
- Calculate a mathematical formula.
- Perform conditional logic.
- Run another map to produce an output.

To enter a map rule, use the rule bar in the Map Editor view in Design Studio. The rule must always starts with an equal sign (=). In the rules, you can build your expression by using any combination of the following items:

- Hard-coded values (literals) including symbols such as <CR><LF> or hexadecimal values such as <<0A>>
- Map functions
- Object names
- Mathematical functions
- Concatenation of multiple objects
- Functional map references
- Brackets and operators
  - Arithmetic operators (+, -, *, /)
  - Logical operators (&, !, |, ^, =)
  - Comparison operators (=, >, <,>=, <=, !=)
- Comments enclosed in /* */, which are ignored during result evaluation

In the mapping rules, you can use all the input objects and all the output objects that occur either previously on the same output card or on the output cards with lower numbers. These options are possible because the output data is produced sequentially one card after another. The object names are usually long. Therefore, you must always use a drag-and-drop method to put the object name in your formula instead of typing it manually.

Design Studio provides a wide selection of functions that you can use in the mapping rules. It contains the following functions among others:

- Bit manipulation and testing functions
- Conversion functions
- Date and time functions
- Error handling functions
- Inspection functions
- Logical functions
- Lookup and reference functions
- Math and statistics functions
- Text functions
- XML functions

You can write your own functions and attach them to Design Studio. You can also call a function in an external library or application or call a method from an external Java class.

With an object of complex type, you can enter the mapping rules either on the group level or on the component level, but never on both levels. If you enter the expression on the group level, the item level cells are disabled, which means that they are unavailable to edit as shown in Figure 3-27.

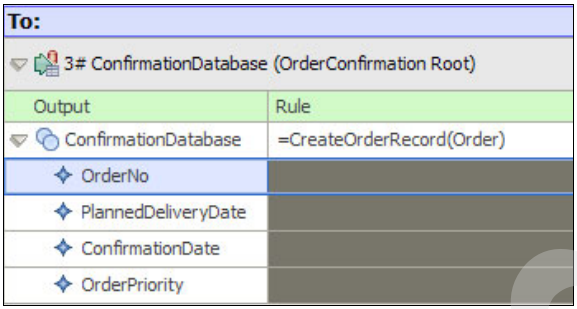

Figure 3-27 Mapping rule entered on the group level

The opposite situation occurs when you start to enter the rule on the item level. In this case, the group level cell becomes disabled as shown in Figure 3-28.

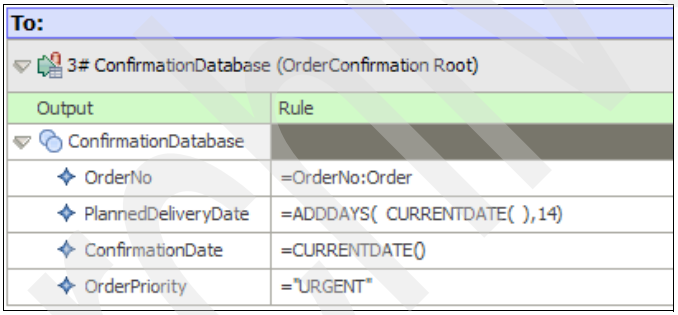

Figure 3-28 Mapping rules entered on item levels

If the maximum occurrence of some objects is greater than one, then in the mapping rules, you must tell WebSphere Transformation Extender how many output objects to produce. There are two possibilities:

- The number of objects to create is known, and you can indicate explicitly how many objects to produce. In such a case, you can use the indexing functionality, which creates an instance of an object (Figure 3-29 on page 81).

In the example in Figure 3-29, we produce exactly two instances of the OrderLine object, by indexing the output.

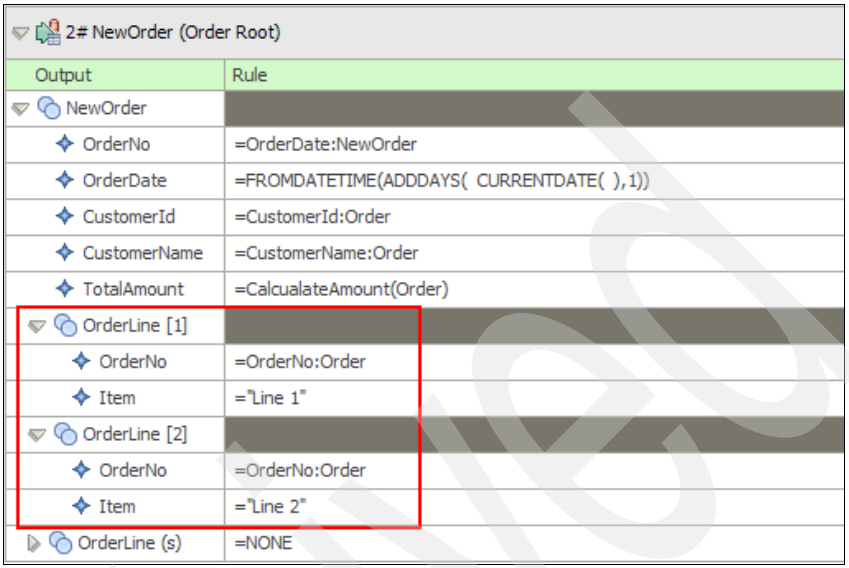

Figure 3-29 Mapping multiple occurrences of an object by using the indexing functionality

- The number of objects to create depends on the occurrences of other objects in the input or output data. In this case, you must create a functional map that converts one object to another for every occurrence (Figure 3-30).

  In the example in Figure 3-30, the OrderLine object in the NewOrder output card is produced for every occurrence of OrderLine from the input card Order.

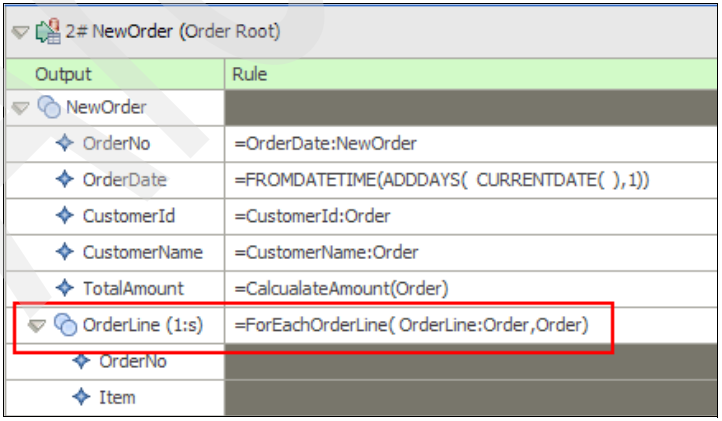

Figure 3-30 Mapping multiple occurrences of an object by using the functional map

3.3.4 Functional maps

A *functional map* is a special map that can be considered as a subroutine that converts one portion of data at a time. We use a functional map in the following situations:

- The output object is displayed more than once.
- The number of occurrences of the output object is not known at the design level and depends on input data.
- The output type is different than the input type.

For example, you want to create a database record for each order line, or you want to prepare a payslip for each person on a given payroll list.

To use a functional map, you must have a *functional map call*. A functional map call is a map rule that is placed on the executable map to call the functional map and parse the portion of data that is required for mapping inside the functional map. It consists of a logical name and then a comma delimited set of arguments inside brackets. Each argument represents a set of data that is needed for mapping the functional map and constitutes an input card in the functional map itself. You will have as many input cards as you have arguments in the functional map call. Consider the following example:

```
=F_Each_Employee (Employee:record:inputfile)
```

The functional map has as many input cards as there are arguments in the functional map call and in the same order. The functional map has only one output card that is typed at the same level as the functional map call in the executable map.

Inside the functional map, you can use only the data that has been passed as arguments to the functional map. For example, if you want to create a database record for each order line, but one of the columns in the database row must be filled with information from the order header, you must also pass this data to the functional map.

In a functional map, you must define one input card for each argument that you want to send to it. The type defined in the input card settings must match the type of object that you send to the functional map. Otherwise, you receive a build error. The same applies to the output card. That is, the type of the output card must be the same as the type of the cell object where the functional map is used.

The functional map looks almost the same as an executable map with the following differences:

- The functional map can have exactly one output card.
- The functional map must have at least one input card.
- The functional map can have source and target settings defined in card settings, but they are not required.

If the functional map has no source or target settings defined, in the Outline view in Design Studio, you can click the icon to see the source or target settings. You cannot build these settings separately.

Inside one functional map, you can call other functional maps. Suppose that you need to prepare labels for Christmas gifts for all children of all employees of all departments in your company. Figure 3-31 presents a possible solution to this problem.

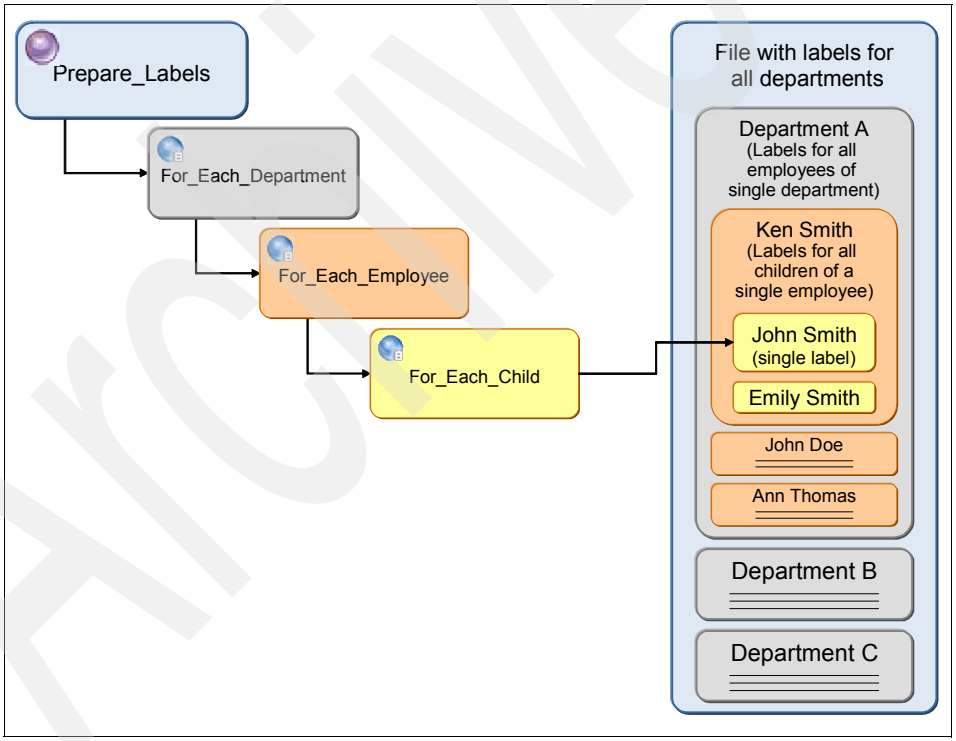

Figure 3-31 Example of using functional maps

The solution follows this sequence:

1. The Prepare_Labels executable map must prepare a file with labels grouped by departments. For each department, it calls the For_Each_Department functional map.
2. The For_Each_Department functional map is called as many times as the number of departments in the input data. This map must prepare labels for each employee who is working in a particular department. It calls the For_Each_Employee functional map.
3. The For_Each_Employee map must prepare labels for each child of an employee. It calls the last map, which is For_Each_Child, once for each child that the employee has.
4. The For_Each_Child map transforms a single child record into a single label.

During compilation of an executable map, all the functional maps that are used are validated. If an error occurs in a rule in any of the referenced maps, the main executable map cannot be built. In the Composition view in Design Studio (Figure 3-32), you can see how all the maps are related to each other.

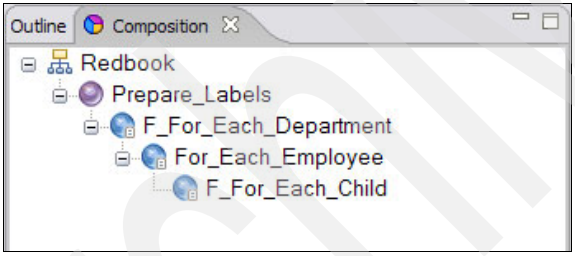

Figure 3-32 The Composition view of the maps

Note: You can use a functional map as a normal executable map. In such case, you must define the data settings for input and output cards. Then the map can be built.

3.3.5 Validation maps

A validation map is probably the first map you will create. Whenever you are using a manually generated type tree (that is a type tree that was not automatically generated by an importer or the Database Interface Designer), create a validation map before you use the type tree in a transformation map. A validation map is a simple map that ensures that the metadata (the type tree) matches the data stream (the physical data) that it is supposed to represent. It compares each data object against the data type that is declared in the type tree.

There are two kinds of validation maps: the quick validation map and the classic validation map. A *quick validation map* contains only an input card and no output cards. A *classic validation map* has one input card and one output card, which has the exact same definition as the input card.

To create a validation map, you must first either create a new map source, or open an existing map source and create a new map. Then you add an input card in the map. The schema refers to the type tree that you want to validate, typed at the top-level element of your type tree. You choose the file adapter (default) and select the input file that corresponds to your type tree (a test input data file that matches your metadata).

In a quick validation map, you do not assign an output card, but immediately build (compile) and run the map. If your type tree corresponds to your data, you get a successful run (`return execution code 0, map completed successfully`). If the run is not successful, activate the trace on input and check the trace file to determine the problem. See Chapter 7, “Troubleshooting” on page 443.

In the classic validation map, you copy the input card and paste it as an output card. Remember to change the file name on the output card. Otherwise, your input file will be overridden. Then you drag the top-level element of the input card to the top-level element of the output card. A good practice is to leave the tree structures of the source and the target cards collapsed. By collapsing the structures, the only field you can map is the top-level element (Figure 3-33).

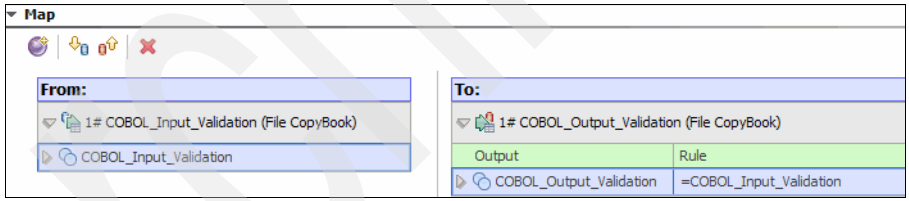

Figure 3-33 Classic validation map

You can then build and run the map. Make sure that the map completes successfully (`return execution code 0`). Then check to see that the map generated as many output objects as there are input objects (Figure 3-34 on page 86). Sometimes it is possible that the objects count differs. For example, when empty tag elements are present in an XML input, they are not generated in the output.

Finally, view the run result, and physically compare the two files. You can also use a file comparison tool. See 8.4.1, “Creating the type trees” on page 504, for an example and further details.

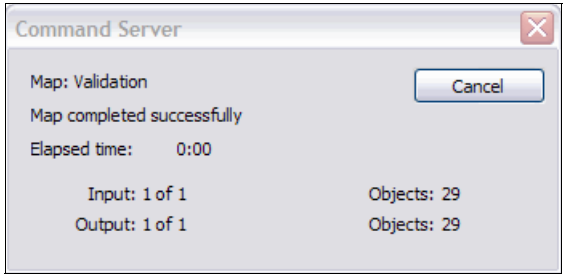

Figure 3-34 Validation map completing successfully

Collapsing and expanding the cards view: Remember that for mapping into a validation map, always collapse the cards view to see only the top-level element of each card. By collapsing the view, the card configuration leaves no other choice than to map the top level input object to the top level output object.

For any transformation map, always expand the cards view to show all data objects of the structures. By expanding the cards view, the output card rule column shows all white cells as mappable objects, where dark grey cells represent the object that we cannot directly map at this stage (groups, multiple occurrence components, and so on).

3.4 Systems

A *system* is a set of maps or subsystems that are linked together into a logical unit of transformation. A *subsystem* is a system that is defined and then used as a component of another system. To keep the system manageable, you might want to create small systems and then include them as a subsystem into a larger system.

A system can include components that reference executable maps and components that reference other systems.

3.4.1 System components

System components are linked upon the decision of the system developer. They do not have to share resources, but they do so frequently. For example, a map generates an output that is used as the input of another map. You can follow the

data workflow through various steps of transformations. Figure 3-35 shows an example of a system and its components.

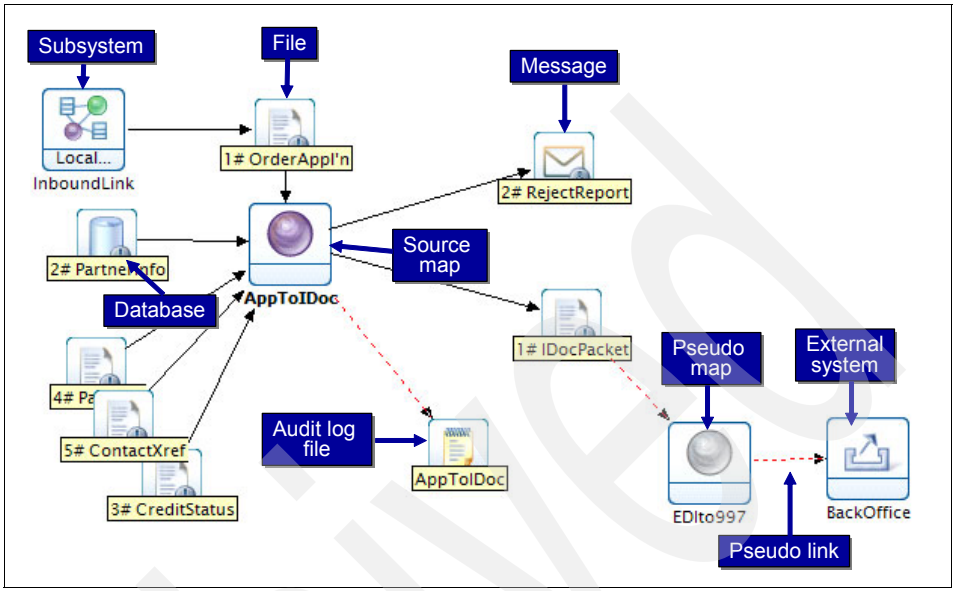

Figure 3-35 System components

You can also have business logic among system components. For example, a map deals with a new customer order and checks if the customer exists. It routes the data to other maps that deal with the order preparation, the payment, and the delivery. The map then calls a subsystem that deals with logistics. All of these maps are components of an Order Process system and do not have to share resources.

A *map component* in a system represents an executable map that has five basic characteristics:

- Name
- Class
- Document link
- Execution settings
- Source and destination interfaces

The Integration Flow Designer treats each map component as a unique object. The characteristics are maintained separately for each map component that you add to a system diagram.

There are three map classes:

- A *source map* references an executable map that is defined within a WebSphere Transformation Extender map source file (a .mms file).
- A *compiled map* references an executable map in a compiled file format (a .mmc file).
- A *pseudo map* acts as a placeholder for a map that has not yet been completely defined.

A *link* represents a connection between two components. It shows the direction of data flow among components at run time. That is, it presents the source and target dependencies between system components.

There are four kinds of links:

- Internal links

  Internal links exist when the source and target are an integral part of the map component. These links are automatically derived and displayed by Integration Flow Designer. They cannot be deleted and are represented by a solid black line between the map components or a dotted red line to audit maps.

- External links

  External links are automatically derived and displayed by Integration Flow Designer. They cannot be deleted. They link a source component that generates data that is used as a data source by a destination component. It can connect: map to map, map to subsystem, or subsystem to subsystem. External links are represented by a solid directed line.

- Pseudo links

  Pseudo links are manually defined. They represent data flow relationships between two components that cannot be derived. They link a map to another map, a map to a subsystem, or a subsystem to another subsystem. These links can be deleted and are represented by a dotted red line.

  Pseudo links help to visualize data flow relationships that are established at execution time. Integration Flow Designer does not embed pseudo links that you define into the process control information that drives WebSphere Transformation Extender servers.

  Pseudo links serve for documentation purposes only.

- Map document links

  Map document links document the system design. You can link any document type that is registered with Windows. Document links assist you in documenting your design. You can view and edit a document by referencing it through the map component to which it is linked.

When you view the document, Windows opens it by using the appropriate application. For example, if the file has a .ppt extension, the file is displayed in Windows by using Microsoft PowerPoint®.

A *subsystem* represents a defined system. By creating a subsystem, you can reuse your systems. A subsystem has four basic characteristics:

- Name
- Document link
- Execution settings
- Source/Destination interfaces

Each subsystem component name *must* be unique. The Integration Flow Designer treats each subsystem component as a unique instance. Subsystem characteristics are maintained separately for each subsystem component.

Systems are designed in the Integration Flow Designer. (See 5.3.7, “Integration Flow Designer” on page 280.) They can be either event driven, by using the Launcher, or command driven, by using the command server and batch or shell scripts, or JCL.

3.4.2 Preparing systems to run

After you design your systems and verify the component relationships, use Integration Flow Designer to prepare them to run. Figure 3-36 illustrates the steps to prepare a system for production. The steps are the same regardless of the server type.

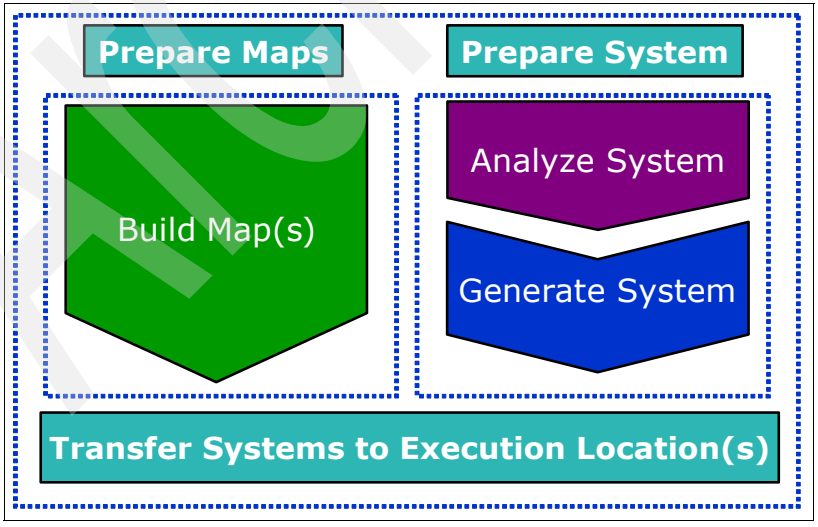

Figure 3-36 Steps to prepare a system for production

To prepare your maps to run, build the compiled maps from any source maps that are referenced in your system.

To prepare your systems to run:

1. Analyze the system for consistency and completeness. If the execution mode is set to Launcher, analysis ensures that all map components have at least one event trigger.
2. Generate the Launcher control files based on the system execution mode.
3. Transfer the systems to the server.

A system is driven by a system launcher file (.msl extension) that is going to be interpreted by the Launcher when it starts. This file contains all information about the system properties, the event that should trigger each map of the system to be executed, and so on. At startup, the Launcher browses all system launcher files that are declared in the administration mode (in the Launcher Administration Java window; see 4.3.2, “Using WebSphere Transformation Extender with Launcher” on page 106). The Launcher uses the Event Manager to initiate each map run based on declared triggers. The Resource Manager synchronizes shared data and timing interfaces among heterogeneous sources and destinations.

3.4.3 Launcher

The Launcher runs systems of maps that are created and generated by using the Integration Flow Designer. These systems of maps that are generated specifically to run in the Launcher are called *system files* (.msl extension). They are sometimes referred to as *Launcher control files*. When the Launcher starts running, it is initialized with .msl files in the deployment directory.

Each platform that has a WebSphere Transformation Extender Launcher installed has a Launcher Administration application. This application allows configuration of secure access rights to monitor and control individual WebSphere Transformation Extender systems.

From the Launcher Administration application, systems can be configured to start automatically, connection ports for the Management Console can be configured, and resource resolution directives can be specified globally or on a system-by-system basis.

Figure 3-37 shows the Launcher methodology.

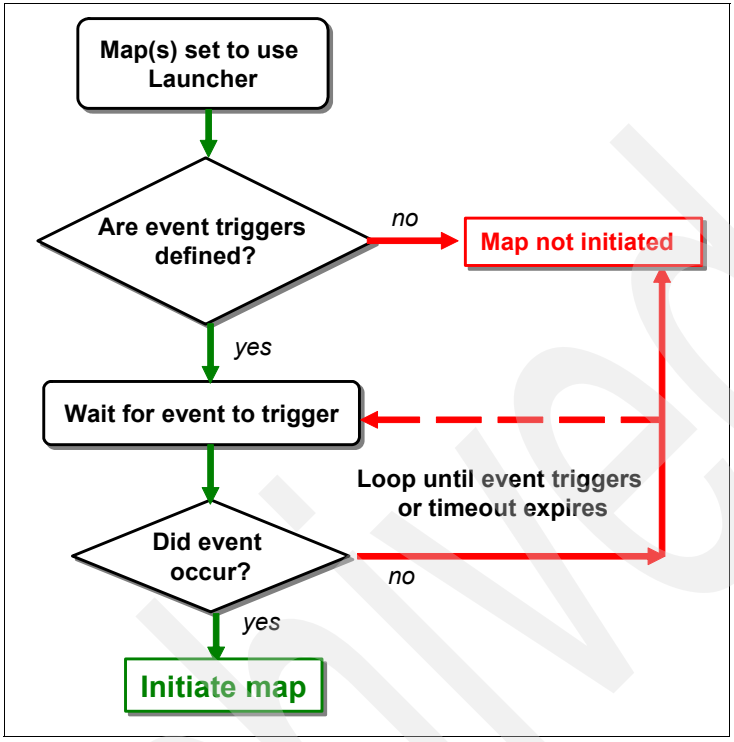

Figure 3-37 Launcher methodology

Map components are run based on either a time event, an input event, or a combination of multiple events. If a map component is not assigned an event trigger, the Launcher does not initiate that map to run.

The Event Manager is the Launcher component that drives the initiation of maps based on subscribed sources. Whenever Launcher detects a new event trigger, the watches that are associated with that trigger have one of the following conditions:

- They are initiated.
- They are placed in an initiation pending state.
- They are maintained in an initiation pending state until all specified event triggers have occurred.

Figure 3-38 shows the default architecture of the Launcher service or daemon.

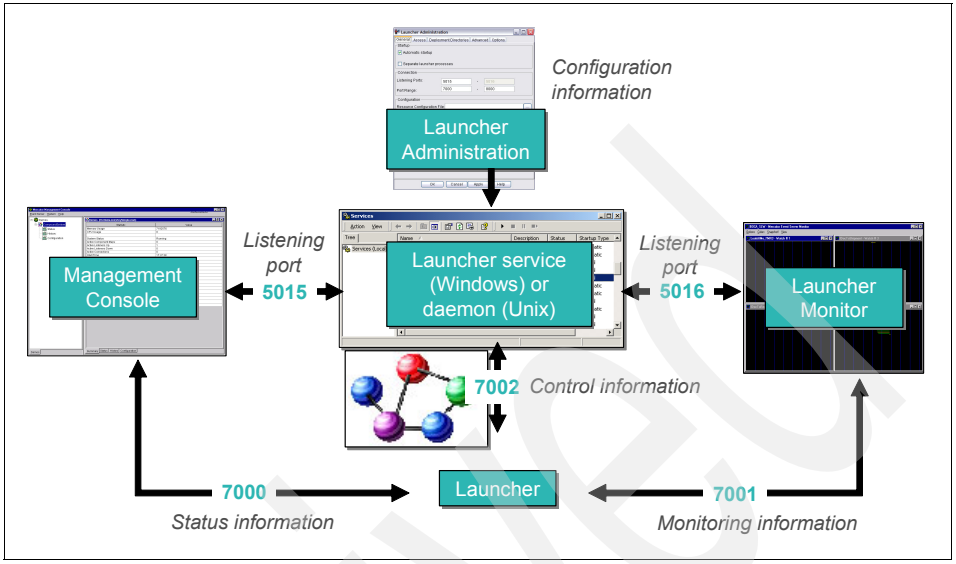

Figure 3-38 Launcher architecture and default port settings

The *Launcher Administration* application contains control settings for the Launcher service or daemon. It includes automatic startup, port usage, users, and deployment directories.

The *Launcher service* is a Java application that is responsible for the coordination of the Launcher and management tools.

The *Launcher* is the process that is responsible for triggering maps.

Listening ports are the ports that listen for incoming requests. By default, the Management Console connects to the Launcher service on port 5015. It sends commands, such as connect, start, and stop to the Launcher service. In turn, the Launcher service relays these commands to the Launcher on port 7002.

The *Launcher Monitor* obtains the Launcher port from the Launcher service over port 5016 (by default). It connects to the Launcher port (7001) to receive monitoring information.

4

Methods of execution

In this chapter, we describe the various methods of execution of a transformation within each edition of WebSphere Transformation Extender.

4.1 WebSphere Transformation Extender runtime editions

WebSphere Transformation Extender is a product that is available with different runtime architectures, depending on the method in which you want to call the transformation runtime.

You can access the transformation engine in the following different ways:

- In Batch mode, in which the map is started with a command
- In Event-driven mode, in which the map automatically starts based on one or more events
- By using WebSphere Message Broker, which starts the map to delegate complex mapping and validation tasks to WebSphere Transformation Extender
- By WebSphere ESB or WebSphere Process Server, both of which can hand off complex transformations and validations to WebSphere Transformation Extender
- From other software by using one of the WebSphere Transformation Extender APIs

  Any program written in one of the supported languages can delegate transformations and validations to WebSphere Transformation Extender by using the APIs.

Figure 4-1 shows a schematic overview of the different execution methods. Each WebSphere Transformation Extender edition is a subset of these methods.

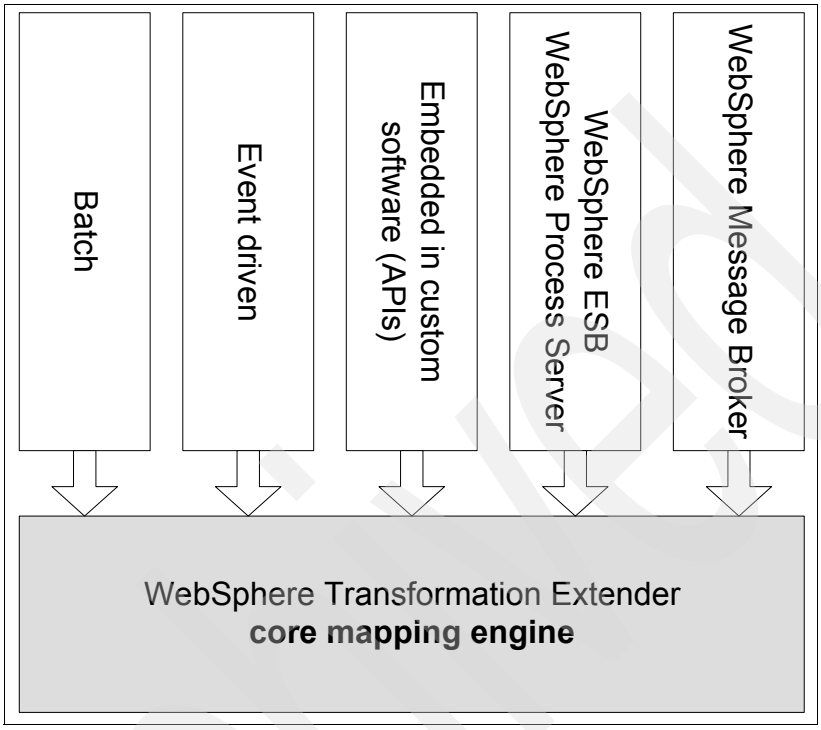

Figure 4-1 Ways to access the WebSphere Transformation Extender run time

4.2 WebSphere Transformation Extender with Command Server

The WebSphere Transformation Extender with Command Server edition is suited to run maps in batch mode. With this edition, you can use a command to run maps from a command line on any system. This command can be called from batch scripts or job schedulers.

In the following section, we discuss how maps can be run in batch mode by using the Command Server to access the transformation engine. Figure 4-2 highlights the upcoming focus on using the batch approach for transformations within the context of all the different ways to access the transformation.

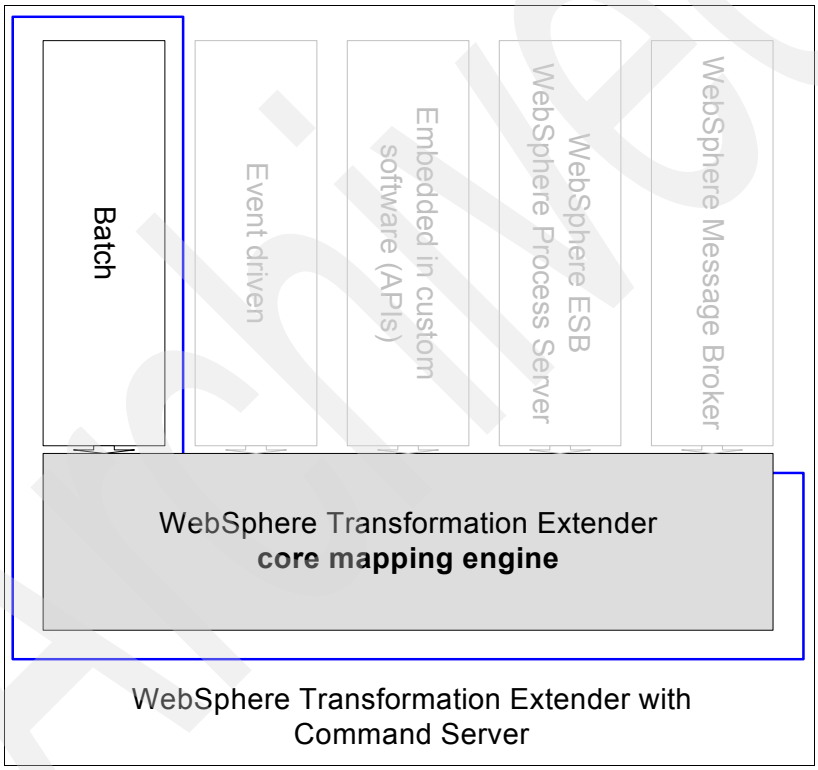

Figure 4-2 WebSphere Transformation Extender with Command Server

4.2.1 Installing WebSphere Transformation Extender with Command Server on distributed systems

The installation process of WebSphere Transformation Extender with Command Server is straightforward. On a Windows machine, you start the `setup.exe` file and click through the installation wizard. Figure 4-3 shows the software packages that you are likely to install on both the development machines and the runtime servers.

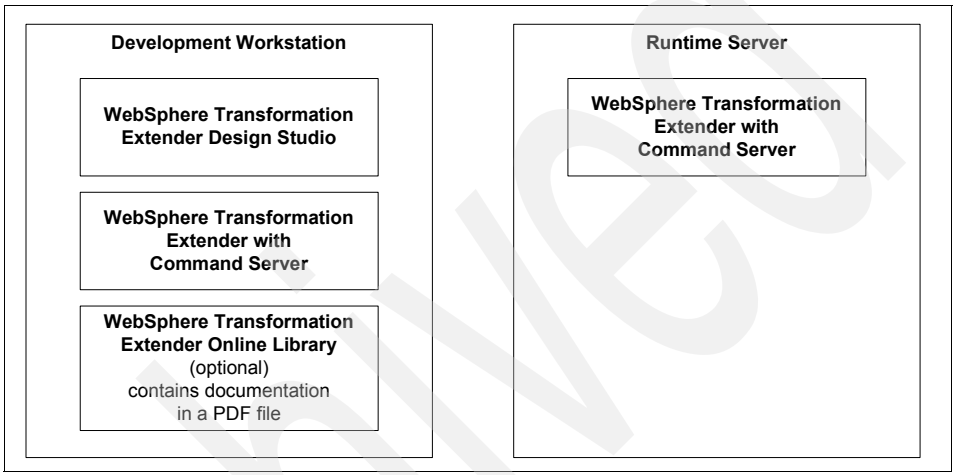

Figure 4-3 WebSphere Transformation Extender with Command Server

In addition to other files, the installation of WebSphere Transformation Extender with Command Server creates an executable file with which you can start a map. This file is in the directory where you installed WebSphere Transformation Extender.

On a Windows system, use the `dtxcmdsv.exe` command file. On UNIX and Linux systems, you use the `dtxcmdsv` command file.

Backward compatibility: For backward compatibility with earlier releases, you can use the `dstx.exe` and `mercnt.exe` files, which are copies of the `dtxcmdsv.exe` file. On UNIX and Linux systems, these files are called `dstx` and `mercator`.

4.2.2 Using WebSphere Transformation Extender with Command Server on distributed systems

You can use WebSphere Transformation Extender with Command Server on distributed systems, which include Windows-based systems and UNIX- and Linux-based systems.

The map command

To run a map, enter the **dtxcmdsv** command with the name of the compiled map file as the first parameter.

For example, assume that you are running a Windows system with the following details:

- WebSphere Transformation Extender is installed in `C:\IBM\WTX8.2`.
- The name of the compiled map file is `master.mmc`.
- The compiled map is in the `c:\maps` folder.

Example 4-1 shows the command to run the map on Windows.

Example 4-1 Running a map on Windows from a command line

```
C:\IBM\WTX8.2\dtxcmdsv.exe C:\maps\master.mmc
```

In another example, assume that you are running an AIX system with the following details:

- WebSphere Transformation Extender is installed in `/opt/IBM/WTX8.2/bin`.
- The compiled map is named `master.aix`.
- The compiled map is in the `/home/wtx` folder.

Example 4-2 shows the command to run the map on AIX.

Example 4-2 Running a map on AIX from a command line

```
/opt/IBM/WTX8.2/bin/dtxcmdsv /home/wtx/master.aix
```

Adding execution commands

You can add execution commands as parameters to specify specific run options. With execution commands, you can perform the following tasks among others:

- Override input sources
- Override output targets
- Control the creation of audit log and trace files

You can find the full list of execution commands in the Online Library or in the WebSphere Transformation Extender Information Center at the following address:

http://publib.boulder.ibm.com/infocenter/wtxdoc/v8r2m0/index.jsp?topic=/com.ibm.websphere.dtx.execcmd.doc/references/r_execution_commands_List_of_Execution_Commands.htm

For example, to send the data of the first output card of the `master.mmc` map to a `c:\output.txt` file, you change the command shown in Example 4-1 on page 98 to the command shown in Example 4-3.

Example 4-3 A map with an overridden output card

```
C:\IBM\WTX8.2\dtxcmdsv.exe C:\maps\master.mmc -OF1 C:\output.txt
```

Note: With execution commands, you can change parameters for the adapters that are connected to a card and change the adapter type. For example, suppose that you have a map that is compiled to send output to a file. If you run the map with a parameter -OM, you can send the output as a message on a queue instead of to a file.

Executing the command

When you enter the map command and press the Enter key, the map starts running and the following actions occur depending on your system:

- On UNIX and Linux systems, the panel is updated to show the status of the running map. After completion of the map, the overall map status is shown. Figure 4-4 shows an example for a map running on Linux.

```
Map file: /products/dstx8101/examples/general/map/sinkmap/sinkmap.mmc
In # 2:       88  Out # 3:       22  Time:    0:00
Map completed (0):  Map completed successfully (0.099 seconds)


[root@pi-erhel30-01 sinkmap]#
```

Figure 4-4 Running a map on Linux

- On Windows systems, a window opens that shows the progress and, after completion, the overall status (Figure 4-5).

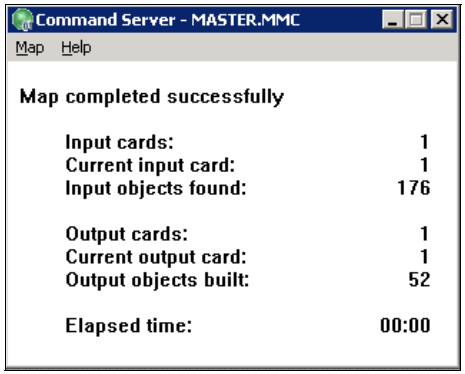

Figure 4-5 The result of running a map on Windows

Including the map command in a script or a scheduler

Typically, you seldom run a map manually from the command line. A common practice is to create scripts that contain the map execution commands and run these scripts, or to include the commands in scheduled tasks.

On Windows systems, you typically create .bat or .cmd files. On UNIX- and Linux-based systems, you typically create .ksh or .sh files.

You can check the map return code in your script and take appropriate actions if the return code does not equal zero. For the full list of all return codes, consult the “Map execution error and warning messages” article in the WebSphere Transformation Extender Design Studio Information Center at the following address:

http://publib.boulder.ibm.com/infocenter/wtxdoc/v8r2m0/index.jsp?topic=/com.ibm.websphere.dtx.md.doc/references/r_map_design_Map_Execution_Error_and_Warning_Messages.htm

Permitting blanks in the path names on Windows

Even though the best practice is to avoid blanks in path names, you might be unable to avoid them in some situations. To work with these blanks, choose one of the following options depending on the location of the blank:

- If a blank is in the path name of the command server executable, wrap the path with double quotation marks (" ").
- If a blank is in the path name of the executable map file, wrap the map with single quotation marks (' ').

Example 4-4 shows the full command syntax.

Example 4-4 Syntax for allowing blanks on Windows

```
"[WTX_Install_path]\dtxcmdsv.exe" '[map_path]\[map_name]' [execution
commands]
```

If you adapt Example 4-3 on page 99 to allow spaces in the path names, you see the results shown in Example 4-5.

Example 4-5 A command on Windows with spaces in the path names

```
“C:\Program Files\IBM\WTX8.2\dtxcmdsv.exe” ‘C:\Documents and
Settings\Administrator\My Documents\maps\master.mmc’ -OF1 C:\output.txt
```

4.2.3 Running a map in WebSphere Transformation Extender with Command Server on z/OS systems

You can use WebSphere Transformation Extender with Command Server on a z/OS system.

Note: Using WebSphere Transformation Extender on Linux on System z is identical to any Linux system because a genuine Linux system is installed.

The WebSphere Transformation Extender maps for z/OS and all of its subsystems are generated specifically for z/OS. The file has the .mvs extension. To build a map for z/OS, in the Map Editor menu of WebSphere Transformation Extender Design Studio, you select **Map** → **Build for Specific Platform** → **IBM z/OS**.

Using WebSphere Transformation Extender on z/OS UNIX System Services is similar to using it on other UNIX systems. There are some specific topics because of the environment, such as direct access to existing host files, or because of the EBCDIC native page code.

Before using WebSphere Transformation Extender on z/OS UNIX System Services, you must create an execution environment and set all environment variables. You create this environment by using the provided JCL, DTXINST, and a **setup** script. The command line is identical to the WebSphere Transformation Extender for UNIX command line, with the same keys words and options. WebSphere Transformation Extender is installed in the `/u/wtxuser/WTX8.2/bin` directory.

In Example 4-6, the compiled map has the `master.mvs` file name, and the compiled map is in the `/u/wtx` directory. This example also shows the command to run the map on UNIX System Services from the command line with the option to override the output card.

Example 4-6 Running a map from the UNIX System Services command line

```
/u/wtxuser/WTX8.2/bin/dtxcmdsv /u/wtx/master.mvs -OF1
/u/mywtx/data/output.txt
```

Using WebSphere Transformation Extender Command Server on a z/OS batch environment requires you to use a job control language (JCL) to execute the environment. See Chapter 6, “WebSphere Transformation Extender on System z” on page 353, for specific information about file names, end of record features, and command line options.

When using WebSphere Transformation Extender Command Server in the CICS environment, you can launch the Command Server in the following ways:

- From a clear terminal by using the DSTX transaction
- From a user program by starting the DSTX transaction in the EXEC CICS START command
- From a trigger transient data queue by initiating the DSTX transaction
- By calling the DSTXCICS program through an EXEC CICS LINK

Command line options are identical to the z/OS batch environment. See Chapter 6, “WebSphere Transformation Extender on System z” on page 353, to learn more about these options.

4.3 WebSphere Transformation Extender with Launcher

The WebSphere Transformation Extender with Launcher edition includes the command line and script execution capabilities of the Command Server, and introduces the Launcher. The Launcher is a stand-alone, event-driven, multi-threaded transformation engine.

The Launcher has similar but less extensive capabilities than IBM WebSphere Message Broker. However, the Launcher’s event server activates transformations (map executions) when triggered by an event, which might be a file creation, a message arriving on a queue, a database trigger, or a scheduler. It is capable of handling complex events, supporting time-based events and availability-based events, and supporting synchronizations on combinations of triggers on multiple input sources.

By using multi-threading, multiple instances of a map can run concurrently, provided that there are no conflicts in resources (for example, files that are used by the maps). This increases the amount of data that can be transformed at one time by allowing multiple transactions to process concurrently. Depending on the platform you use, the Launcher can run as a multi-threaded application (daemon) (UNIX, Linux, or Linux on System z) or service (Windows).

The Launcher runs on a specified server and can manage one or more systems. (See 3.4.1, “System components” on page 86). Systems can be run as individual processes or grouped together and run as a single process (compound system).

Distributed systems require one Launcher per server machine.

Figure 4-6 illustrates how WebSphere Transformation Extender with Launcher edition includes both batch processing and event-driven processing capabilities. The Command Server allows for command line and script execution, while the Launcher is a stand-alone, event driven, multi-threaded transformation engine.

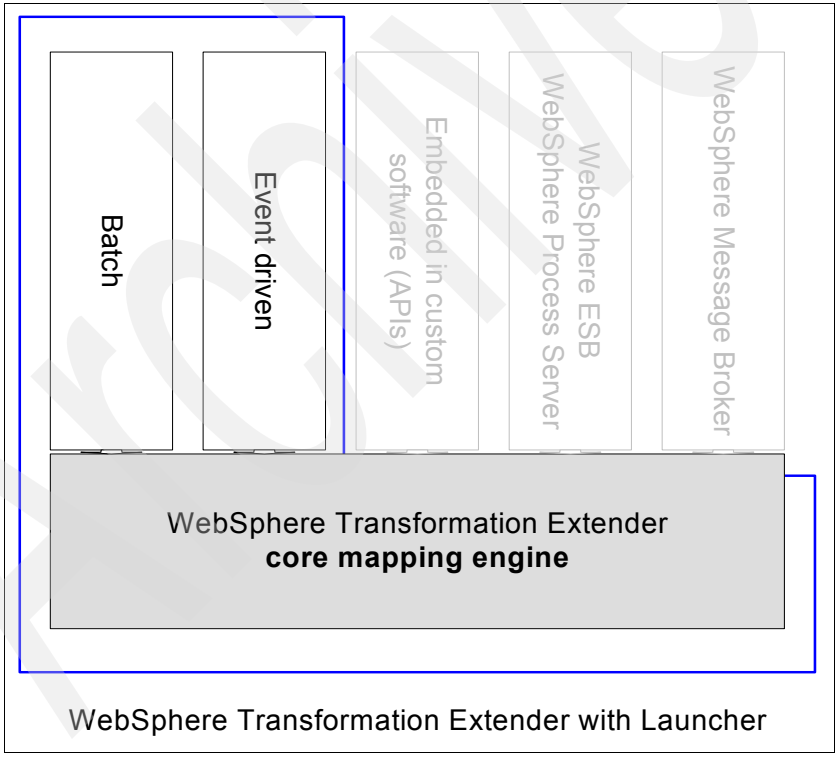

Figure 4-6 WebSphere Transformation Extender with Launcher

4.3.1 Installing WebSphere Transformation Extender with Launcher

Figure 4-7 shows the software packages that you are likely to install on the development machines, the runtime servers, and optionally a separate monitoring system. Installing IBM WebSphere Transformation Extender with Launcher is straightforward.

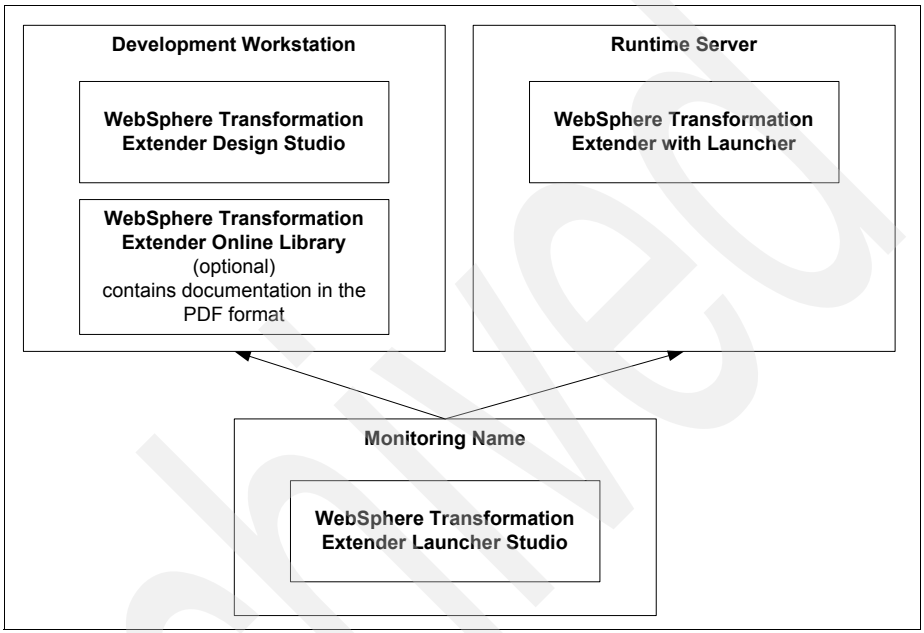

Figure 4-7 Installation of WebSphere Transformation Extender with Launcher

To install WebSphere Transformation Extender with Launcher:

1. Start the **`setup.exe`** program.
2. In the Choose Setup Language window (Figure 4-8), select the appropriate language and click **OK**.

Figure 4-8 Choosing the setup installation language

Installation language: The installation language that you choose does not influence the default setting of the software language. This is based on your regional settings.

3. Review and accept the license agreement. Then enter your name and the company that owns the license and click **Next**.
4. In the next window, choose **Typical** installation.

   By choosing Custom installation, you can decide which adapters and administration tools to install, and whether you want to include Clustering Support and OBDC drivers as shown in Figure 4-9. On Microsoft Windows, the installer might ask you to reboot (depending on the operating system version).

   In this example, we click **Next** to accept the default installation method.

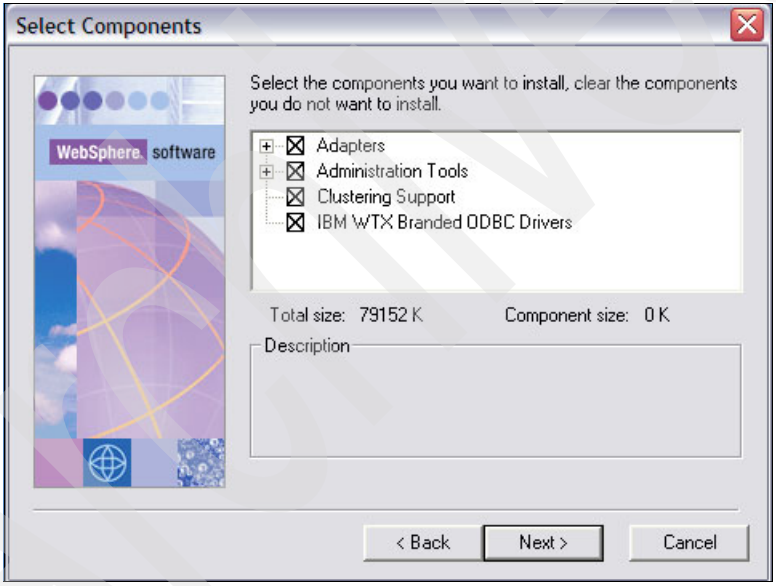

Figure 4-9 Custom installation select component window

After the installation completes, a new Launcher folder is in the Start menu of IBM WebSphere Transformation Extender 8.2 (Figure 4-10).

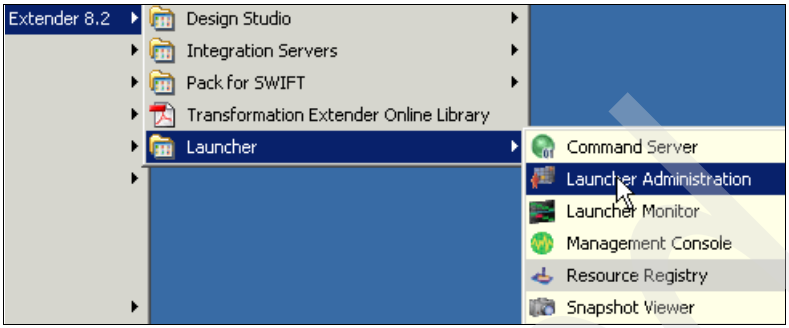

Figure 4-10 Launcher folder in WebSphere Transformation Extender

You also see that a WebSphere Transformation Extender 8.2 Launcher service is available in the Services console (Figure 4-11).

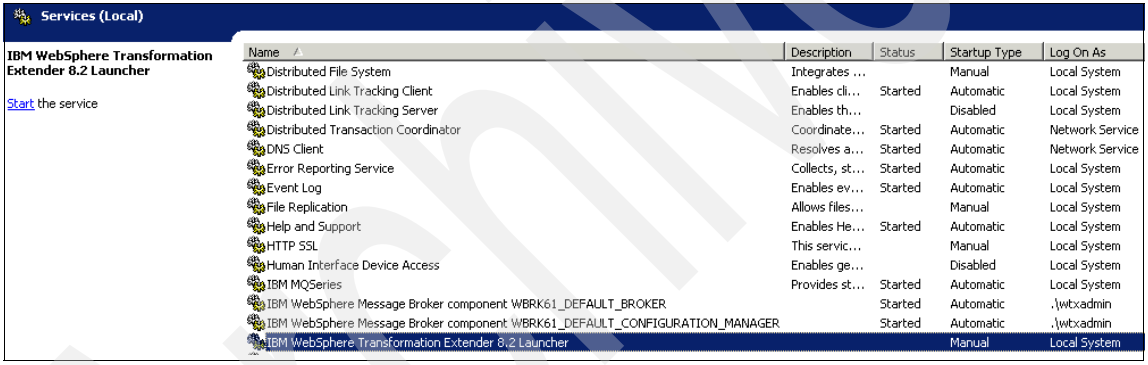

Figure 4-11 WebSphere Transformation Extender Launcher Service

4.3.2 Using WebSphere Transformation Extender with Launcher

You develop the systems that run on the Launcher in the Integration Flow Designer. See 5.3.7, “Integration Flow Designer” on page 280.

On the run time, you must prepare the daemon or service for execution. First modify the `dtx.ini` file to generate log files for the Launcher run time. Then start the Launcher Administration tool to prepare for execution, declare the systems to be run, grant access for control and monitoring, and specify the run mode and options. After doing these administration tasks, start the daemon or service, and start the systems (in case they are not configured to start automatically when the

daemon or service starts). After you start the daemon or service, you can use the monitoring tools to monitor the running systems.

The monitoring tools are included in the runtime installation, but they can also be installed separately by using the *Launcher Studio* package. This way we can monitor the runtime remotely or locally.

The *Management Console* is the latest monitor tool. It provides a lot of information about system runs and failures.

By using the *Launcher Monitor*, you can generate snapshots (on demand or for specific cases) that can be viewed by using the Snapshot™ viewer.

Preparing the Launcher run time: Initialization settings of the dtx.ini initialization file

Before going into the Launcher administration, activate the Launcher logging. Uncomment (that is, remove the semicolon) the `LauncherLog=eswc` line in the `dtx.ini` file, which is line 69 in WebSphere Transformation Extender 8.2.0.3, as shown in Example 4-7.

Example 4-7 The dtx.ini LauncherLog section - Removal of the semicolon to activate the log

```
; launcher log types - e = error, w = warning, s = startup c = cfg summary
LauncherLog=ewsc
```

Removing the semicolon generates a log with information about every error, warning, startup, and configuration summary in the `logs` subfolder of our WebSphere Transformation Extender installation.

Preparing the Launcher run time: Launcher administration

Administration: The Launcher administration on Windows, UNIX, or Linux is configurable through a Java-based GUI. For distributed operating systems other than Windows, this administration is done by using an X Window System (X11). On z/OS you can choose between native administration interfaces (the ISPF Panel and the Operator's Console) and user interfaces based on UNIX operating systems (Launcher Administration and Management Console), again based on X11.

The Launcher Administration tool is a Java-based GUI, with which you can configure the Launcher. This tool has five tabs, which we explain in the sections that follow.

The General tab

Figure 4-12 shows the **General** tab of the Launcher Administration.

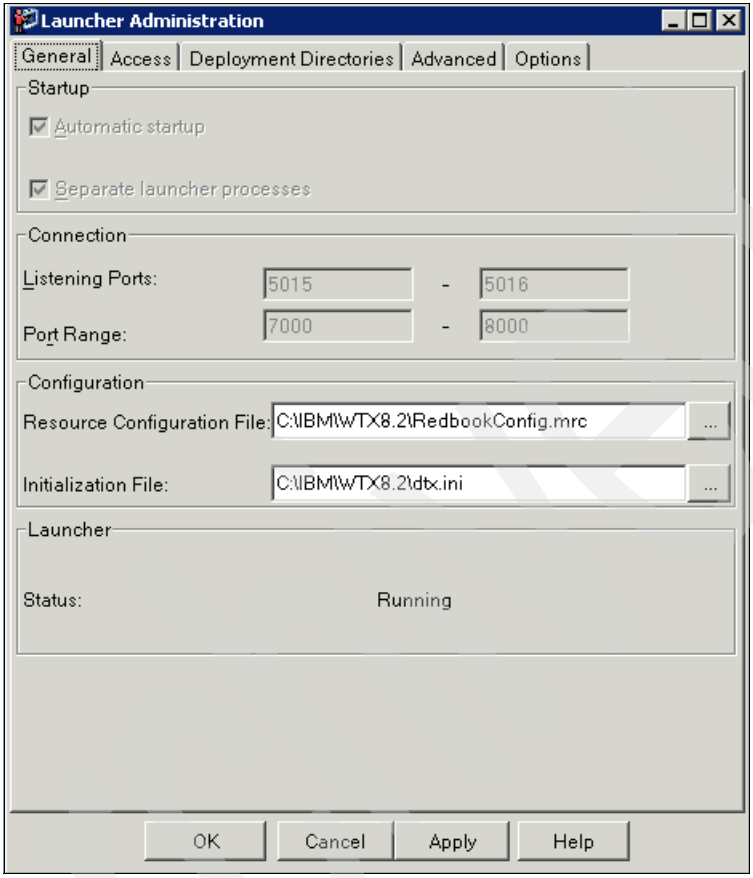

Figure 4-12 The Launcher Administration General tab with a running Launcher

The **General** tab has the following settings:

- With the *Automatic startup* option, systems can start automatically when the service or daemon starts.
- By selecting the *Separate launcher processes* option, you can choose whether you want to manage systems as a whole compound unit (the classic way) or to manage each system separately.
- The *Connection* settings are used to specify which ports will be used by the Launcher.

- Under *Configuration*, you specify the following files:
  - The default *resource configuration file* (.mrc file) that the Launcher should use. Keep in mind that each system, if managed separately, can have its own .mrc file.
  - The default *initialization file* (.ini file) that the Launcher should use. Keep in mind that each system, if managed separately, can have its own .ini file.
- *Status* indicates the current status. When the Launcher is in *Running* status, you can only modify the Configuration settings. The changed settings are only taken into account after a Launcher restart. To change the other settings, the Launcher must be in *Stopped* status (stopped service or daemon).

  Status is not dynamically updated and requires the Launcher Administration tool to be closed and opened again.

The Access tab

On the **Access** tab (Figure 4-13 on page 110), you can configure the User Name, Login Name, and Password settings for the various users. You can also define a set of users. For each user, you can grant or revoke rights to Start/Stop, Pause/Resume, or simply Monitor the systems. User rights apply to all systems. To enable monitoring, you must provide a user, login name, and password, and grant rights to Monitor.

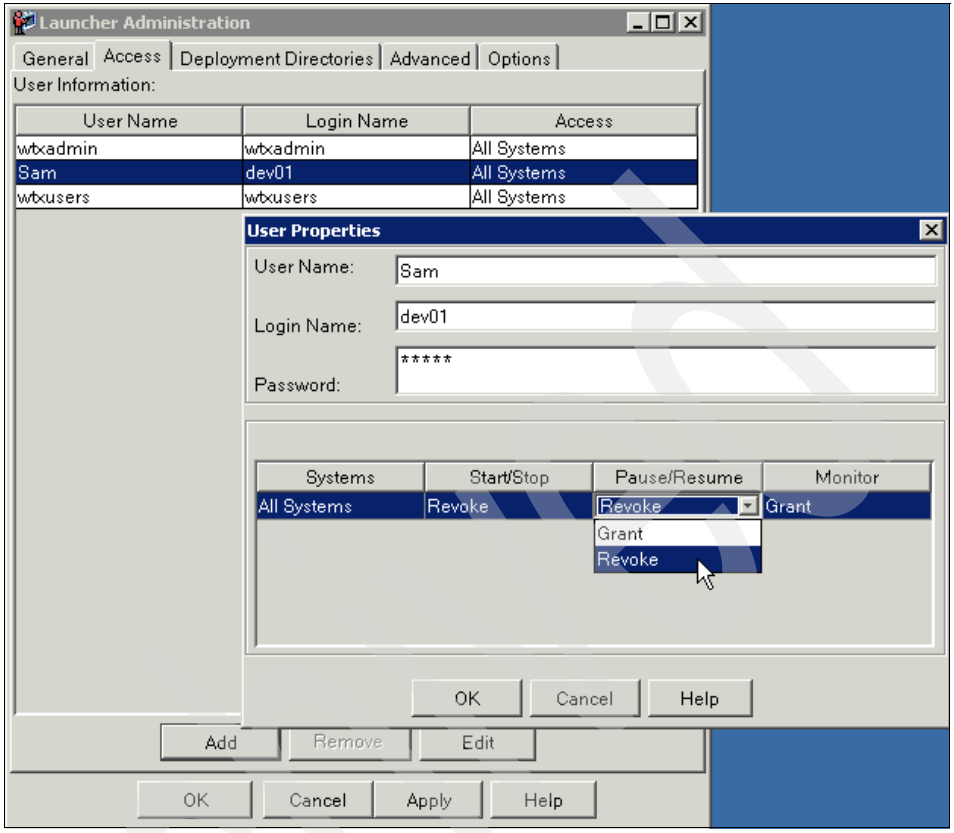

Figure 4-13 Launcher Administration Access tab - Editing user properties

The Deployment Directories tab

You can use the **Deployment Directories** tab (Figure 4-14 on page 111) to configure the directories in which to place system Launcher files (.msl file). At Launcher startup, the Launcher browses each folder and collects .msl file information for each system. The default is the systems subfolder of the WebSphere Transformation Extender installation path.

Temporary subfolder: It is practical to create a subfolder (for example `old`) inside each deployment directory to temporarily place launcher system files (.msl) that we do not want to add to Launcher for several reasons (for example, testing, and temporary disabled systems). Rather than erasing and redeploying the system, we can activate or deactivate a system by moving the Launcher system file to or from this subfolder.

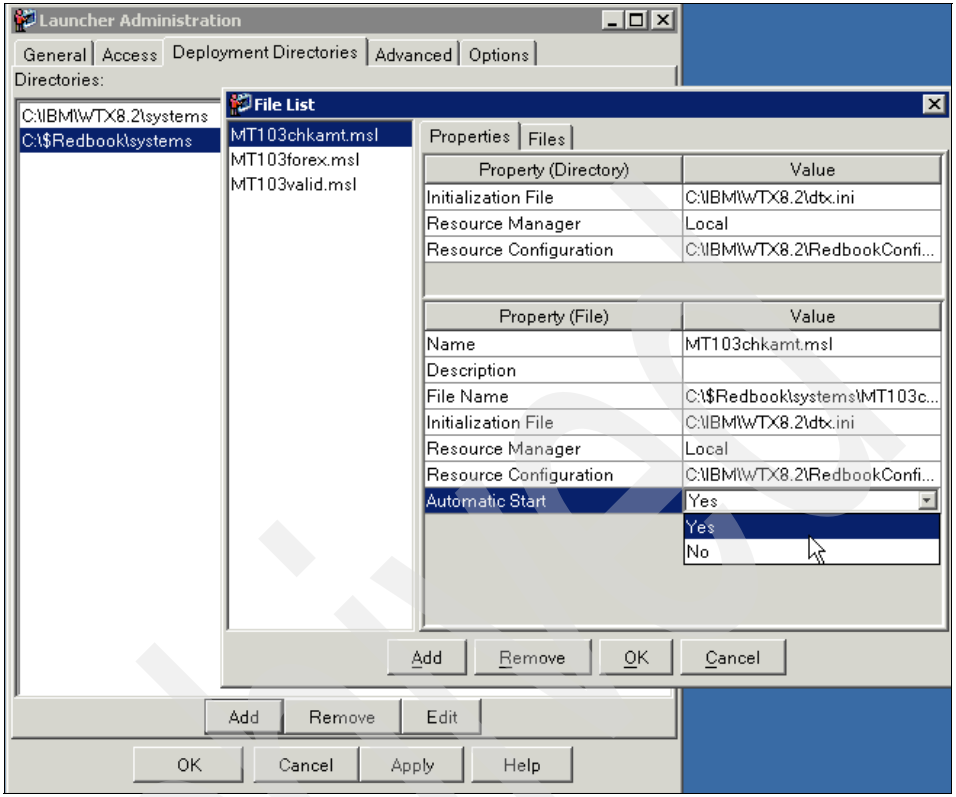

Figure 4-14 Launcher Administration Deployment Directories tab

If the *Separate launcher processes* option is selected on the **General** tab, the Edit button is active. In this case, you must manually add (by clicking **Add**) each system that we want the Launcher to run. If we do not add these systems, a warning message is displayed that indicates that no system is configured to run. Editing the directory opens a File list window in which you can click the Add button to add a new system file.

The Launcher Administration tool then browses for available .msl files in that directory and gives you the opportunity to add the desired launcher files. As shown in Figure 4-15 on page 112, select the launcher system files (.msl files) that you want to add and click **OK**.

Tip: You can press and hold down the Shift key to add several control launcher files at one time.

Figure 4-15 Listing of available launcher system files (.msl files)

After adding Launcher system files, you can edit the properties of each Launcher file such as the name, description, initialization file, resource manager, and resource configuration. If Automatic Startup is selected on the **General** tab, you can also specify the Automatic Start (Yes/No) property. If both options are set to *Yes*, then the current system starts automatically at Launcher start. Otherwise you must start the system manually by using the Management Console when the Launcher is running.

The Advanced and Options tabs

You use the **Advanced** tab to declare a Java Remote Method Invocation (RMI) server host name for clustering purposes.

You can use the **Options** tab to define Firewall Server/Client Ports and the font size.

The Launcher Management Console setting

Before starting the Launcher, you must configure the Management Console settings to start or stop, pause or resume, and monitor our systems:

1. Select **Management Console** in the Launcher or Launcher Studio start menu.
2. From the Java GUI, select **Launcher** → **New** to declare a new Launcher run time.

In the New Launcher window (Figure 4-16), you can configure a logical name, a host name (host name or IP address), and a port number (#5015 default value).

Logical name: Choose your logical name carefully. The name declared here is the name that you use in script mode (for status request, and in case of **separate event processes** mode for starting, stopping, pausing and resuming systems).

You can also specify a refresh interval (5 second default to avoid too many requests to the Launcher) and the security's login name and password as defined on the **Access** tab of the Launcher Administration. See also "The Access tab" on page 109.

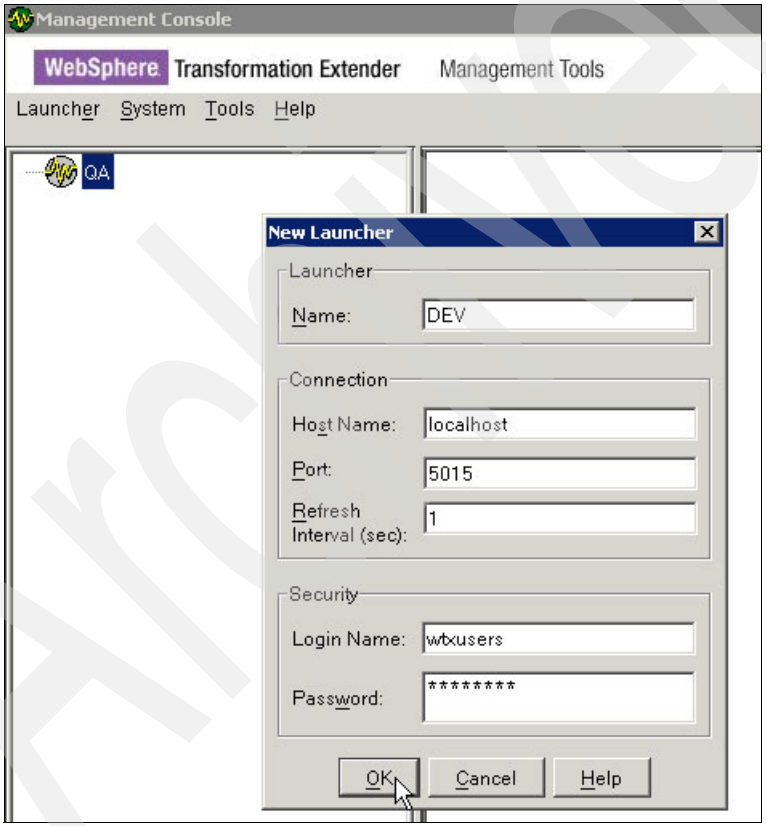

Figure 4-16 Adding a new Launcher on the Launcher Management Console

In the bottom left corner of the Management Console, you see a tab for each declared launcher and below it a Status line (Figure 4-17). This line gives essential information about the state of your request. For example, it shows what the problem is in case of a connection failure to the Launcher.

Figure 4-17 Management Console Launcher Selection tab and Status line

The Launcher run time: Starting and stopping the service

You are now ready to start the Launcher service or daemon. In Windows, select **Start** → **Administrative Tools** → **Services** to open the Services console. From the Services console, right-click the **IBM WebSphere Transformation Extender 8.2 Launcher** service and select **Start** (or click the **Start** link; Figure 4-18).

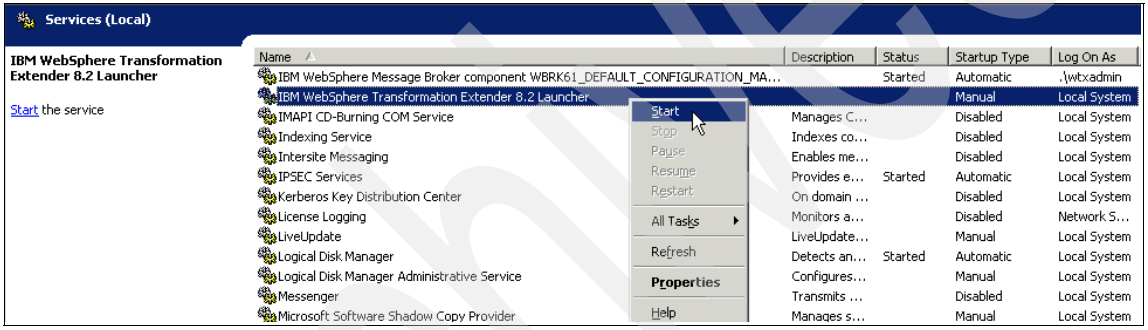

Figure 4-18 Starting the Launcher service

On UNIX or Linux, use the `launcher.sh -start` command to start the daemon.

Batch and shell script commands: Batch or shell script commands are available to start, stop, pause, resume, and get status or summary information from the Launcher daemon. See the product documentation for further details.

The Status column in the Windows Services panel shows a status of *Started*. The default service is set as Manual startup type. Change this setting to Automatic on a production machine, so that the service automatically starts at operating system startup. This service can also be monitored by a high availability system to raise an alert if it is stuck or turned off. The service can also be put into a load balancing or clustering management system (Microsoft Windows Cluster Server, SunCluster, and so on).

You can now connect by using the Management Console tool to monitor and control the Launcher runtime.

Monitoring the Launcher run time with the Management Console

If you declared multiple Launchers on the Management Console, select the appropriate tab on the bottom left corner of the window and click **Connect** (Figure 4-19). The Status line shows *Connecting* and changes to *Done* after the Management Console is connected to the target Launcher.

Figure 4-19 Connecting to the Launcher

Make sure that you watch the information for the correct server and system. The system that is displayed is shown in the title bar of the window on the right side (Figure 4-20). On the left side, you can also see other systems, including systems that are not started and, therefore, cannot be monitored. Remember to check the upper right side of the window.

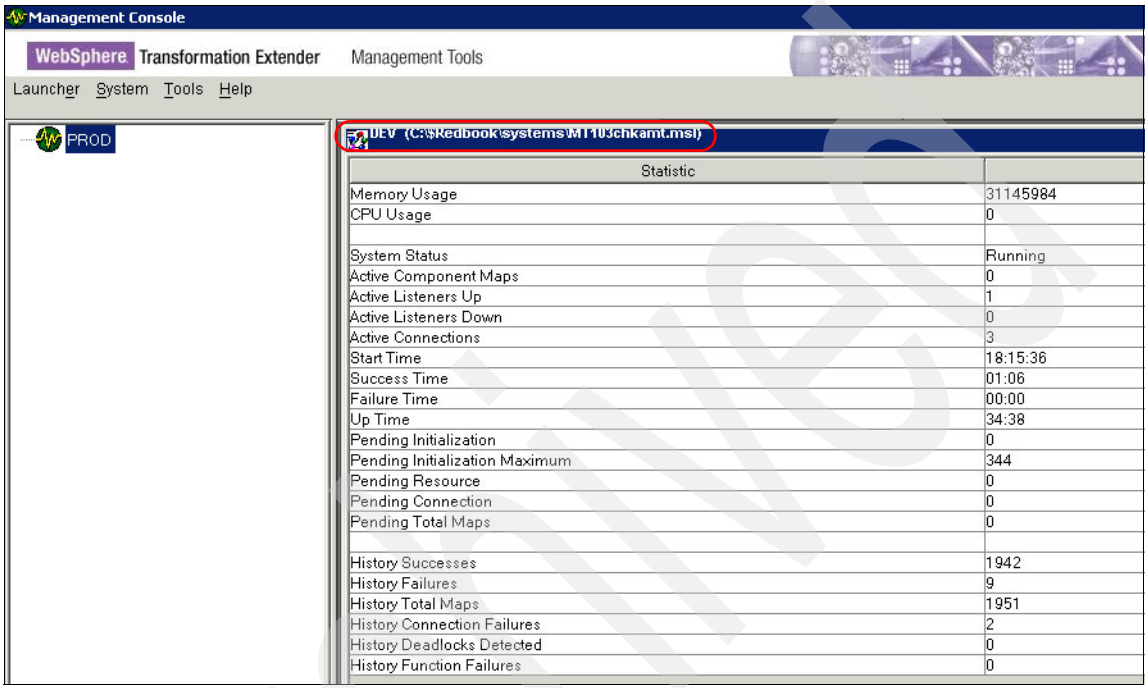

Figure 4-20 Title bar on the right side of the window showing which system you are monitoring

There are four information tabs, which are explained in the following sections. Figure 4-21 on page 118 shows an illustration of these tabs. Note that the summary is provided by selecting the specific .msl file for a specific system.

The Summary tab

The **Summary** tab is the main window and shows the following summary information:

- Current Memory and CPU Usage
- System Status (ready to start, running, paused, or stopped)
- Active Component Maps (maps that are currently running)
- Active Listener Up and Down
- Active Connections (connections to applications that are currently open and active)

- Start Time of the system
- Success Time (time spent running successful maps)
- Failure Time (time spent running a map in failure)
- Up time (time spent since start time; if you add the spent time to the start time, you get the current time)
- Pending Initialization (maps that are currently waiting for initialization) and Pending Initialization Maximum (maximum amount of maps that have been waiting for initialization at a certain time)
- Pending Resource (number of maps that are currently waiting for resources, such as a shared data target)
- Pending connection (number of maps waiting to connect to applications)
- Pending Total Maps (number of maps that are currently pending, sum of pending initialization, resource, or connection)
- History Successes (number of maps that run successfully since the system started)
- History Failures (number of maps that run in failure since the system started)
- History Total Maps (total number of maps that run since the system started, sum of successful and failed runs)
- History Connection Failures (total amount of connections that failed)
- History Deadlocks Detected (number of map failures resulting on deadlocks)
- History Function Failures (number of failures in GET, PUT, RUN, DBLOOKUP, DBQUERY, and EXIT functions)

The Status tab

The **Status** tab (Figure 4-21 on page 118) shows current information about Active Components, Pending Components, and Adapter connections. Each of them has related subtabs.

Active Components shows the maps that are currently running, with the following information:

- Component (name of the watch component)
- System (name of the system)
- State (getting input, validating input, building output, running map “x”, GET or PUT adapter “x”, or ending output)
- Card number (number of the card that is currently active)
- Resource (adapter that is used for the card)

Note: The card number listed is a sequential number. For example, if you have three input cards and two outputs cards, you have a total of five cards, and card 4 refers to output card 1 (the fourth card in sequence).

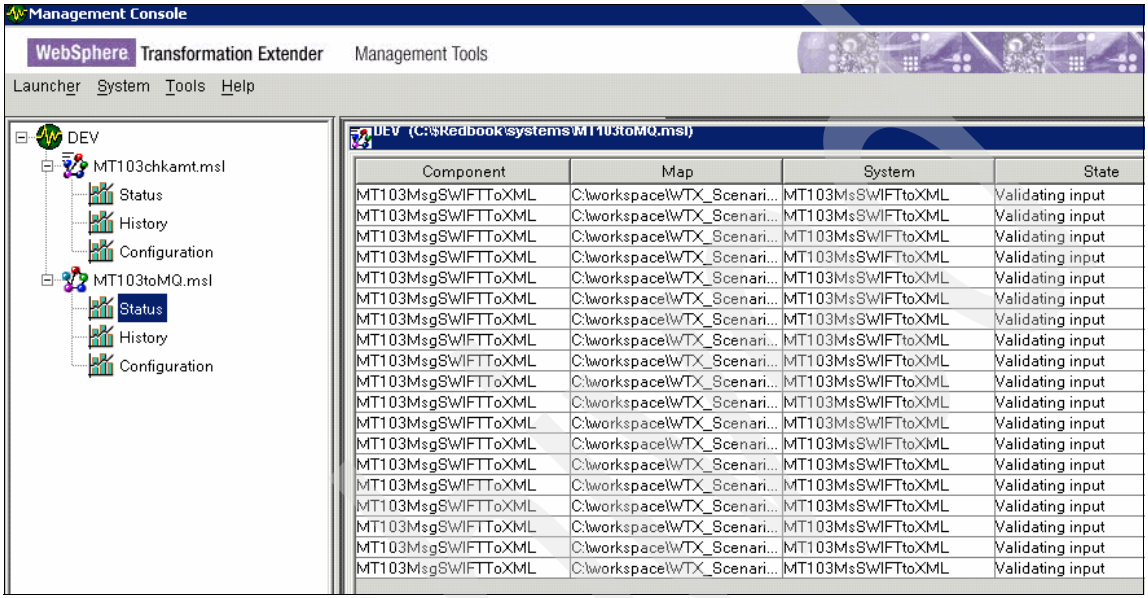

Figure 4-21 Status example from Active Components showing several instances of a multithreaded map

Pending Components shows currently pending maps, with the following information:

- Component (name of the watch component)
- System (name of the system)
- State (resource pending, connection pending, retry pending, or initialization pending)
- Card (number of the card that is currently pending)
- Reason (for resource pending, the reason that the resource is locked)
- Resource Type (resource or connection that is unavailable)
- Resource Name (name of the resource)

Adapter Connections shows current information about adapters:

- Adapter (name of the adapter)
- Open (number of currently open connections)
- Active (number of currently active connections; used by running maps)
- Idle (number of connections that are currently idle)
- Pending (number of currently pending connections)

The History tab

The **History** tab (Figure 4-22) shows historical Map Failures, Adapters connection, and Function Failures. Each of these items has related subtabs.

Map Failures shows historical map failures:

- Component (name of the watch component)
- Map (path and name of the compiled map)
- System (name of the related system)
- Card (number of the card that caused the error)
- Card Error (return code of the adapter)
- Map Error (return code of the map)

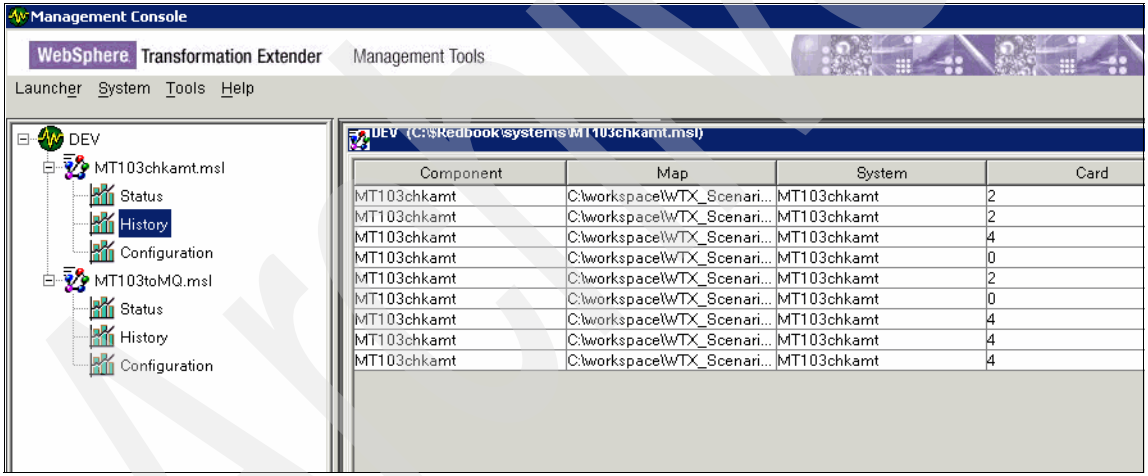

Figure 4-22 History Map Failures showing information about maps that did not completed successfully

Card and map errors: *Card error* refers to the return code that is received by the adapter. It is related to the application error. A *map error* refers to the map execution return codes. In this example, we have Source not available (12), Target not available (9), and one or more inputs was invalid (8).

Adapter Connections shows historical information about adapters (Figure 4-23):

- Adapter (name of the adapter)
- Requests (number of connection requests)
- New (number of created connections)
- Reused (number of times that existing connections are reused)
- Successes (number of successful GETs or PUTs on the adapter)
- Failures (number of failed GETs or PUTs on the adapter)

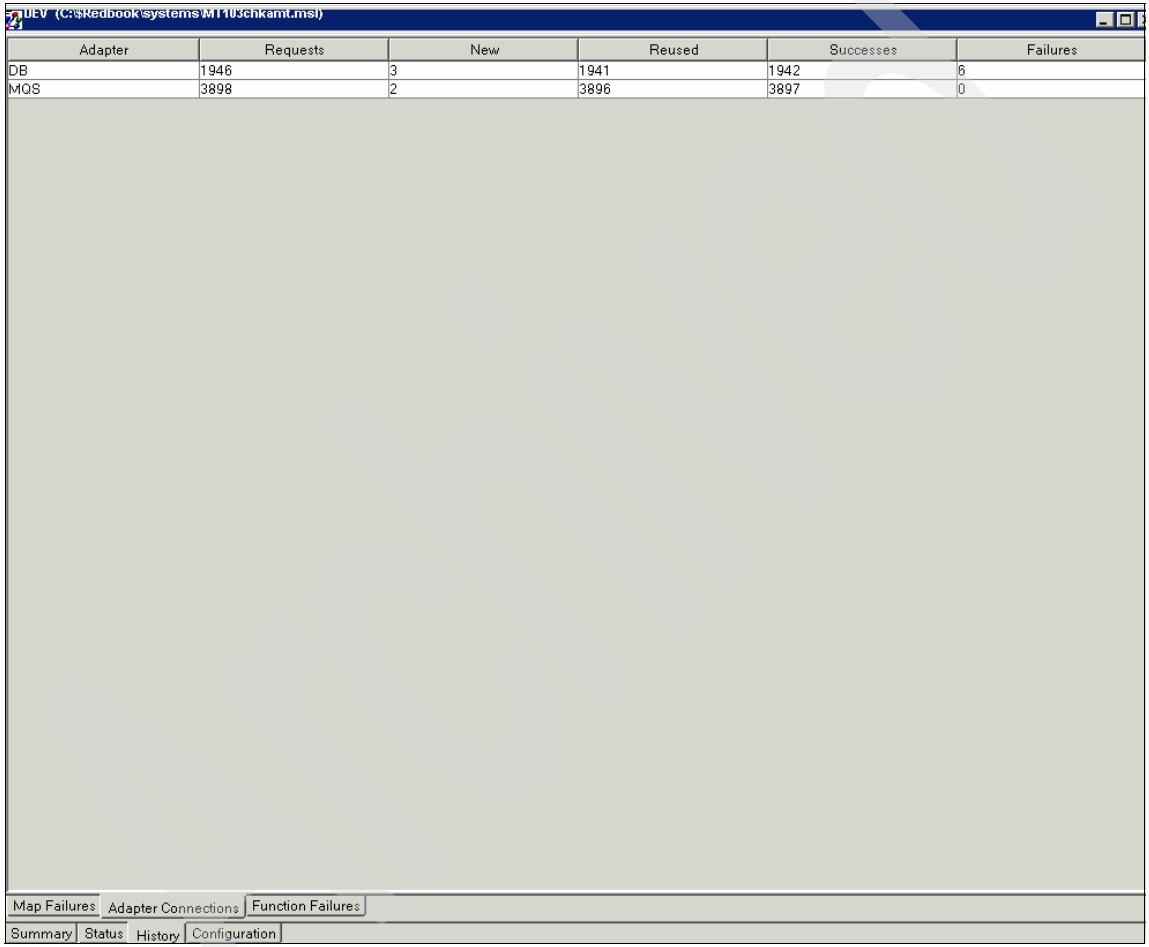

Figure 4-23 Adapter Connections showing historical information

Function Failures shows historical failures that are related to dynamic functions:

- Function Name (name of the function that failed)
- Component (name of the map component that contains the function)
- Argument (map name for the RUN function, adapter for GET, PUT, DBLOOKUP, and DBQUERY)
- Reason (why the function failed)

The Configuration tab

The **Configuration** tab shows the current Launcher configurations for System and Adapters Connections. Each item has related subtabs.

System shows the system configurations:

- Maximum Concurrent maps (maximum maps that can run in parallel)
- Maximum Concurrent map per watch (maximum instances of the same map that can run in parallel)
- Pending Initialization High (user-defined threshold for the number of adapter listeners and time events pending initialization)
- Pending Initialization Low

  After reaching the Pending Initialization High amount, the number of items pending must reach this amount before maps can begin triggering again.

InitPendingHigh and InitPendingLow: You can configure InitPendingHigh and InitPendingLow in the `dtx.ini` file, and you can configure them per system. For example, if InitPendingHigh is set to 10 and InitPendingLow is set to 5, the listener thread pauses processing when the number of occurring InitPendings reaches 10. It resumes processing when the number gets down to 5.

Adapter Connections shows adapter configurations from the `dtx.ini` file:

- Adapter (name of the adapter)
- Idle Time (time after which the connection is expected to be disconnected)
- Keep Time (time during which connection is expected to remain valid and will not be tested)
- Keep Minimum (minimum number of connections to keep)
- Advisory Limit (connection soft limit, which is only checked when opening a connection)

- Mandatory Limit (connection hard limit)

  The total number of connections will not exceed this limit and additional requests will be placed in Connection pending)

Monitoring the Launcher run time with the Launcher Monitor

The Launcher Monitor provides a graphical representation of systems runs, with which you can take a snapshot that can then be analyzed with the Snapshow Viewer for detailed information about the map components run. It is sometimes used to get snapshots on run errors.

In Figure 4-24, the various lines represent several map runs. The blue lateral lines symbolize time. The length of the line symbolizes the time that the map took to run. Several parallel lines mean parallel runs of several instances of the same map. Colors symbolize the return code of the run. If the runs completed successfully, the line is green. The line is yellow if there is a warning, and the line is red if there is an error.

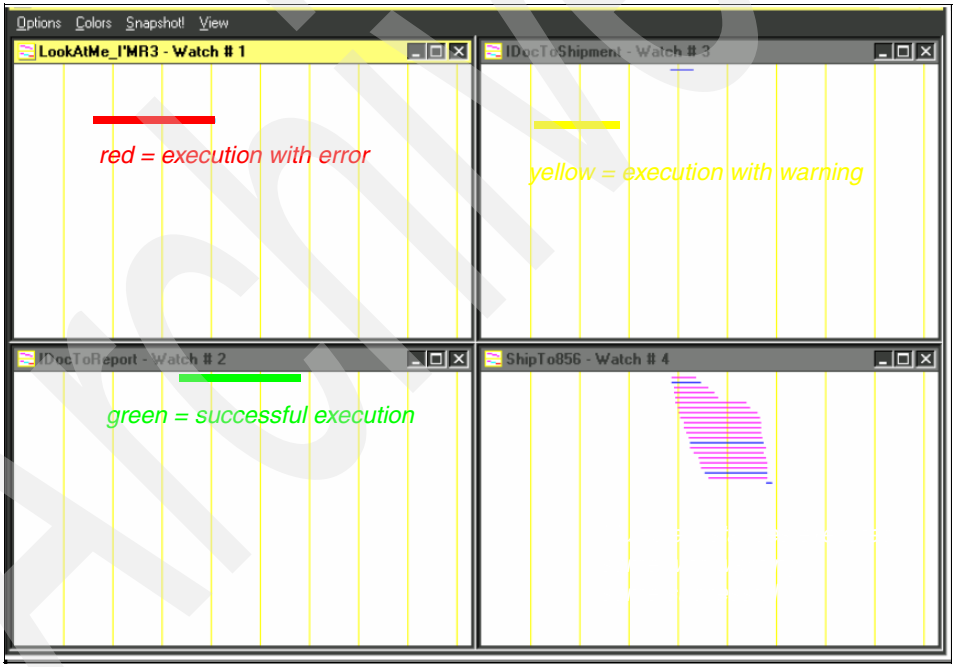

Figure 4-24 Launcher Monitor with four active watches windows

The major benefit of the Launcher Monitor is the ability to take snapshots based on certain criteria:

- On demand only
- On error
- On error or warning
- Continuously

With the Snapshot Settings window (Figure 4-25), you can configure when to take snapshots. If you select the On Demand Only option, the tool waits for you to click the **Snapshot!** button on the menu bar.

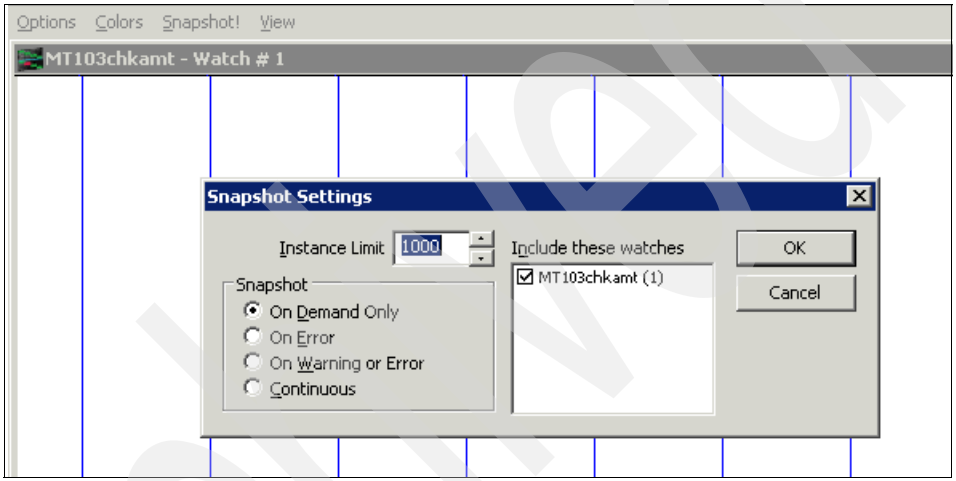

Figure 4-25 Launcher Monitor allow a graphic view to see system runs, per watches

A snapshot generates a .mss file inside the root of WebSphere Transformation Extender installation (one file per snapshot). The .mss files can be opened with the Snapshot Viewer. By right-clicking any instance in the Snapshot Viewer, you see detailed information about the selected map component run, as shown in Figure 4-26.

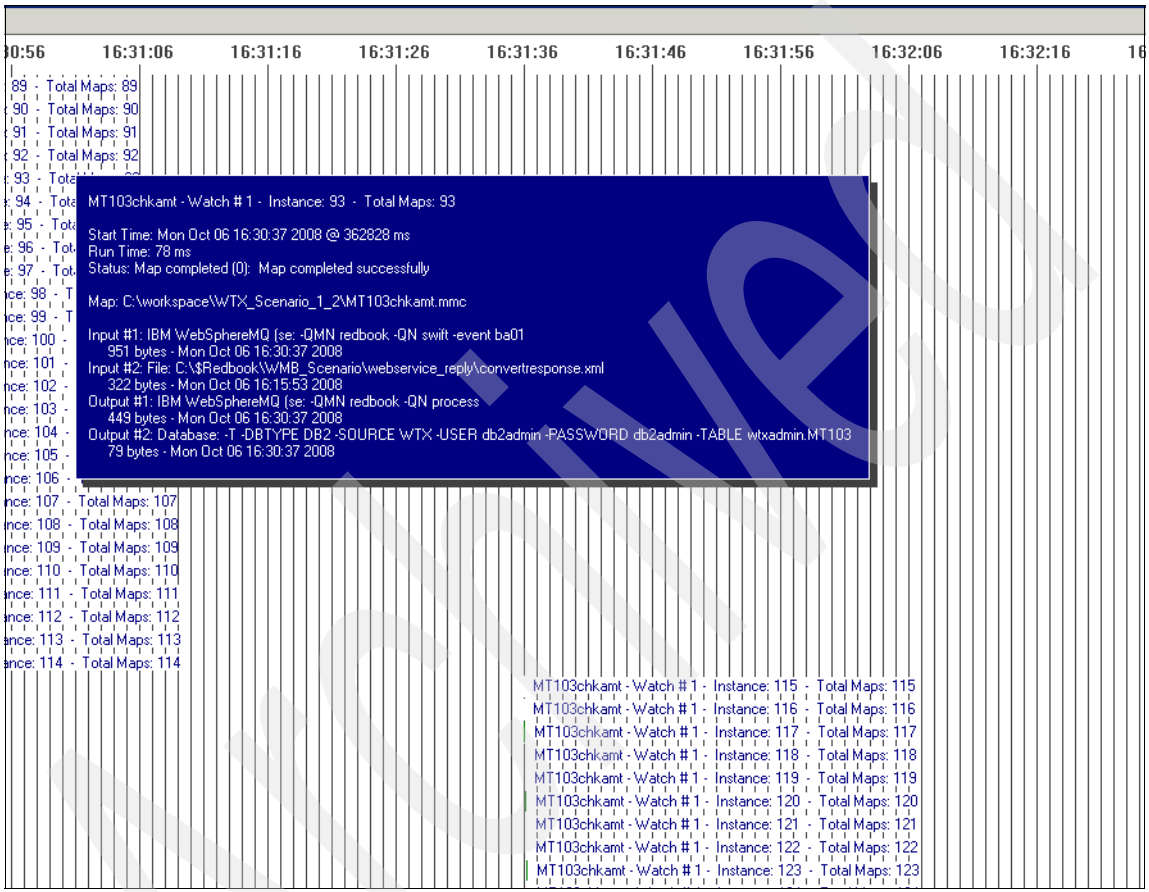

Figure 4-26 Snapshot View showing multiple instances running and their details

Troubleshooting the Launcher

There are several reasons why the Launcher might not start.

No Launcher system file (.msl) is in the deployment directories

Check the startup section of the system log file in the `logs` subdirectory of the installation. The system log file has a name that uses the following convention:

`[system_name}.msl[datetimestamp].txt`

Figure 4-27 shows an error message within the log file indicating the inability to open the .msl file.

```
======================================================================

*** STARTUP:

Status:Failure
Time: Mon Oct 06 22:00:10 2008

Errors:
      Error opening MSL file:
      C:\$Redbook\systems\MT103chkamt.msl

======================================================================
```

Figure 4-27 Error within the log file describing the inability to open the .msl file

One or more maps referenced in the Launcher system file are not found

Check the startup section of the system log file in the `logs` subdirectory of the installation. The system log file has a name that uses the following convention:

```
[system_name}.msl[datetimestamp].txt
```

Figure 4-28 shows an error message within the log file indicating that the map cannot be found.

```
======================================================================

*** STARTUP:

Status:Failure
Time: Mon Oct 06 21:46:30 2008

Errors:
      Map not found:
      C:\workspace\WTX_Scenario_1_2\MT103chkamt.mmc
      Mismatch in card count between MMC
(C:\workspace\WTX_Scenario_1_2\MT103chkamt.mmc) and MSL

      Map not found:
      C:\workspace\WTX_Scenario_1_2\MT103chkamt.mmc

======================================================================
```

Figure 4-28 Error within the log file describing that the map cannot be found

The triggered file source contains a directory path that is invalid or does not exist

This error is difficult to catch because the system will start successfully. We must to examine the system log. The system log file has a name that uses the following convention:

```
[system_name}.msl[datetimestamp].txt
```

Review the trigger section as highlighted in Figure 4-29 to validate that a defined path and real path do not match.

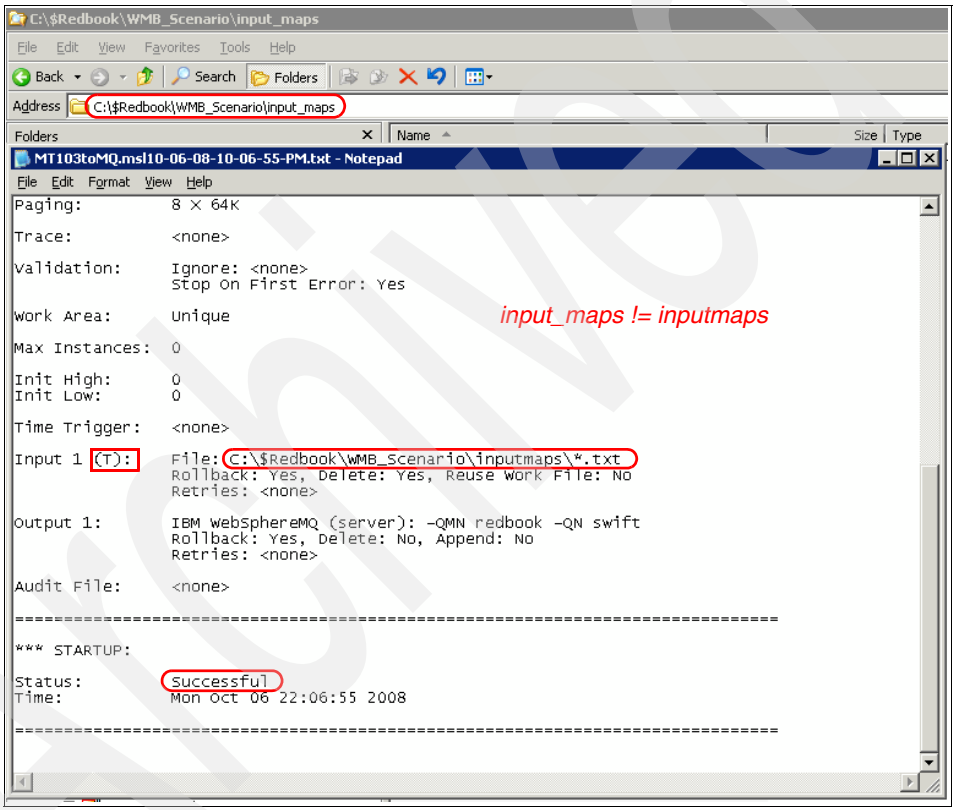

Figure 4-29 Difference of path or file name on a source trigger: Difficult to troubleshoot

On Windows, a triggered file source is on a remote computer, and the Launcher service is set to use 'System Account'

In this case, change the account settings of the Service.

A triggered non-file source uses an adapter that it cannot find

Examine the Launcher logs and check for an adapter initialization failure.

The Launcher starts but then stops running

This problem can be caused by one of the triggers:

- Memory allocation
- Too many threads
- Application connections exceeded
- Inability of third-party software to handle the data volume
- Operating system version issue

4.4 WebSphere Transformation Extender for Integration Servers with WebSphere Message Broker

WebSphere Transformation Extender for Message Broker is a subset of WebSphere Transformation Extender for Interaction Servers, as shown in Figure 4-30.

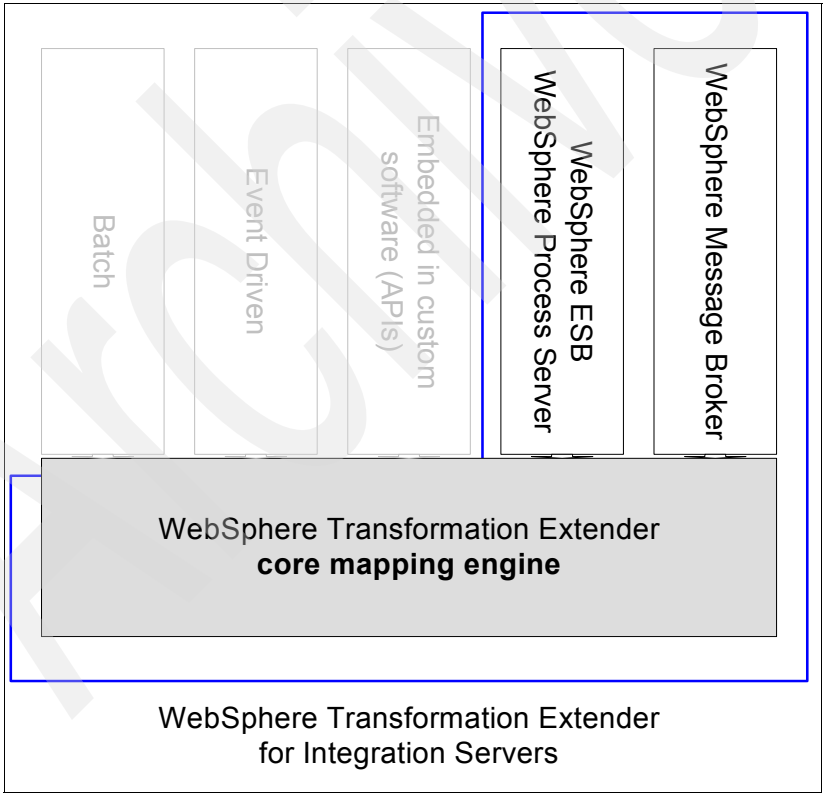

Figure 4-30 WebSphere Transformation Extender for Integration Servers

4.4.1 Installing WebSphere Transformation Extender for Integration Servers with WebSphere Message Broker

Figure 4-31 shows the packages that you install on a development workstation and a runtime server, to use WebSphere Transformation Extender for Message Broker.

- The development workstations require the Message Broker Toolkit so that you can design the message flows. They also require the Transformation Extender Design Studio so that you can create the maps. Then if you install WebSphere Transformation Extender for Integration Servers, it integrates the tooling of WebSphere Message Broker and WebSphere Transformation Extender (so you have one tooling environment). WebSphere Transformation Extender for Integration Servers adds an extra node on the palette of available nodes in WebSphere Message Broker. Optionally, you can also install the Online Library, which gives you the documentation in PDF format.
- The runtime server requires the WebSphere Message Broker runtime and WebSphere Transformation Extender for Integration Servers to provide the broker with the runtime libraries of the WTX Map node.

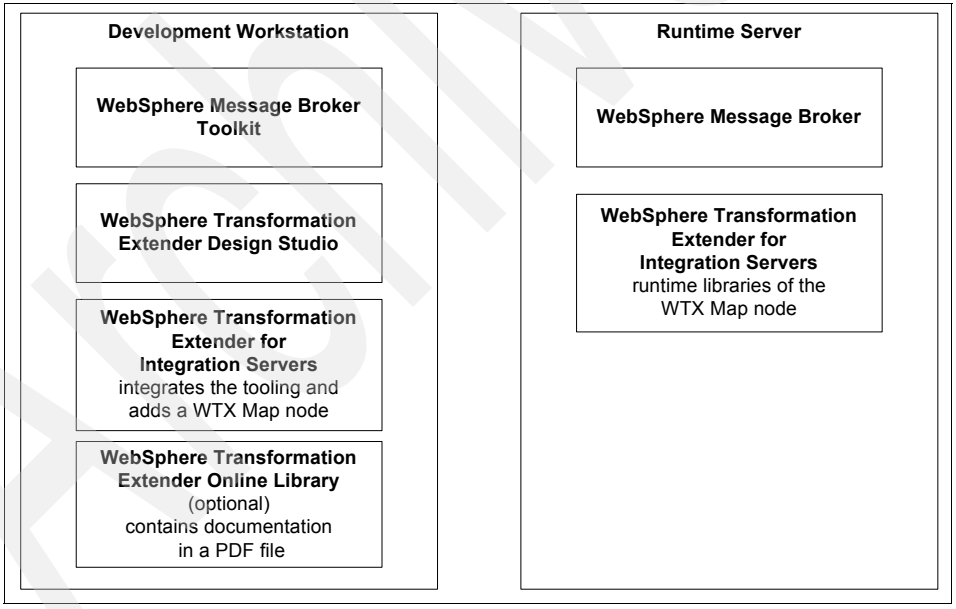

Figure 4-31 WebSphere Transformation Extender for Integration Servers with WebSphere Message Broker

For the scenarios, we installed WebSphere Transformation Extender for Integration Servers on an existing development system with the following products already installed:

- WebSphere Message Broker Toolkit 6.1.0.2
- WebSphere Message Broker 6.1.0.2
- WebSphere Transformation Extender Design Studio 8.2.0.3

To add WebSphere Message Broker for Integration Servers on a Windows platform:

1. Run the `setup.exe` file to start the installation program.
2. Select your setup language (Figure 4-32) and click **OK**.

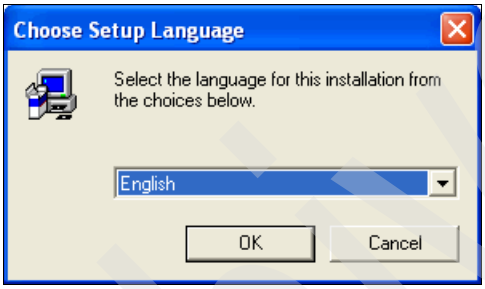

Figure 4-32 Choosing the setup language

3. Click through the windows for the license agreement and user information.

4. In the Setup Type window (Figure 4-33), select a setup type. The Typical setup type includes all components. Click **Next**.

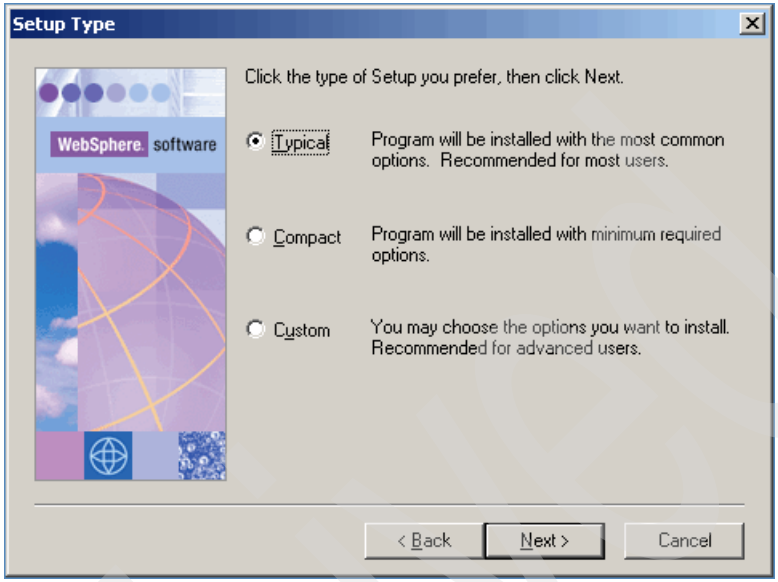

Figure 4-33 Types of setup

5. In the windows that follow, click **Next**.
6. When you see that the installation starts copying files (Figure 4-34), wait until the copying of files has finished.

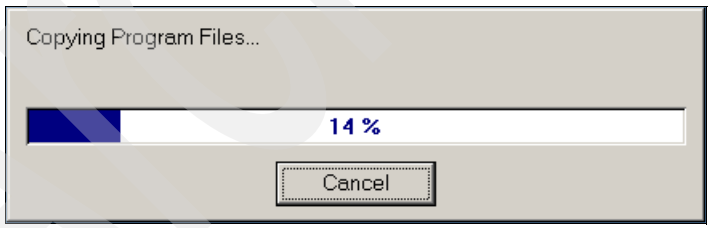

Figure 4-34 Copying program files

7. In the Setup Complete window (Figure 4-35), click **Finish**.

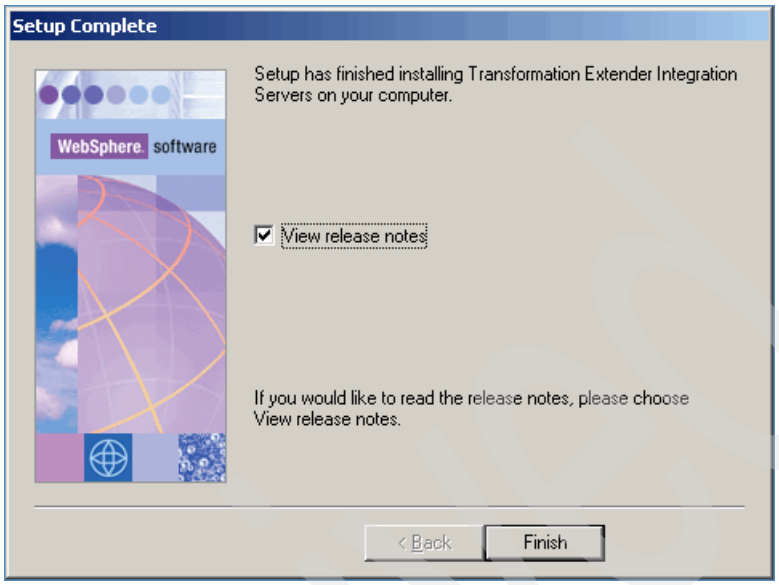

Figure 4-35 Setup complete

WebSphere Transformation Extender for Integration Servers is now installed on your system.

To verify that WebSphere Transformation Extender for Integration Servers is successfully installed on a WebSphere Message Broker development system:

1. Check the integration of the perspectives:

   a. Start the WebSphere Message Broker Toolkit. From the Windows desktop, click **Start** → **WebSphere Message Broker Toolkit 6.1** (Figure 4-36).

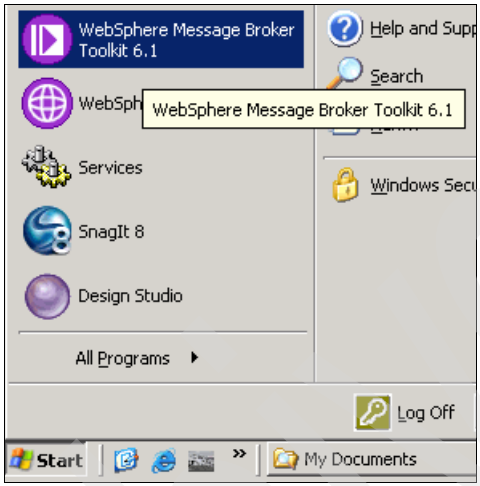

Figure 4-36 Starting the Message Broker toolkit

   b. In the WebSphere Message Broker Toolkit (Figure 4-37), click **Window** → **Open Perspective** → **Other**.

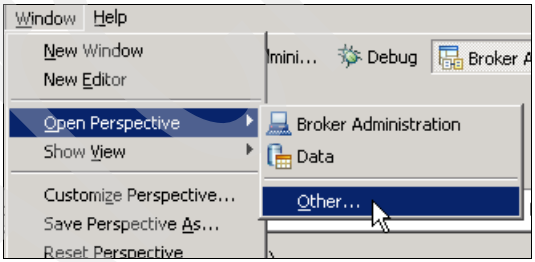

Figure 4-37 Opening another perspective

In the list of available perspectives, you now see the Transformation Extender Development perspective (Figure 4-38).

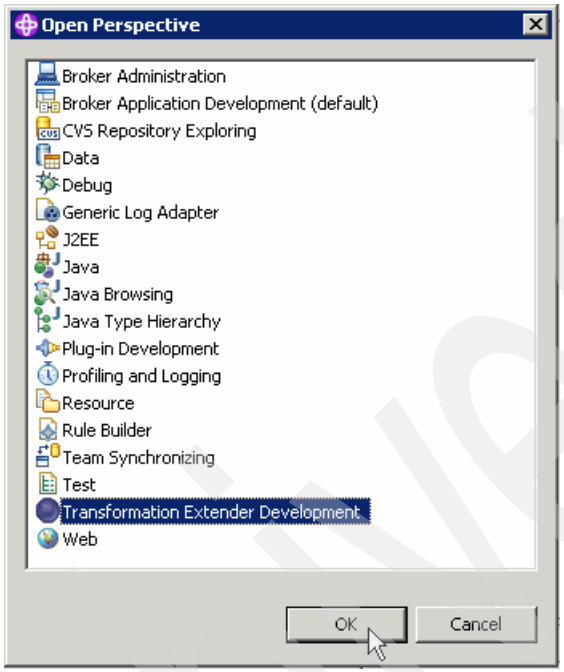

Figure 4-38 The Transformation Extender Development perspective

With the availability of this perspective in the Message Broker Toolkit, you can have a single development environment for both WebSphere Message Broker and WebSphere Transformation Extender.

Note: To add the Transformation Extender Development perspective to other Eclipse installations or if the Transformation Extender Development perspective is not accessible in your WebSphere Message Broker Toolkit:

1. Go to the `<WTXDesignStudio>\wtx_eclipse\eclipse\links` folder.
2. Copy the `wtx_es.link` and `wtx_esdoc.link` files.
3. Paste the files to the `links` folder of your Message Broker Toolkit or other Eclipse installation.

2. Check the availability of the WTX Map node. Create a new Message flow and open the palette of available nodes.

   A new node is added to the palette of WebSphere Message Broker nodes. The node is called a *WTX Map node* and is in the WebSphere TX drawer, as shown in Figure 4-39.

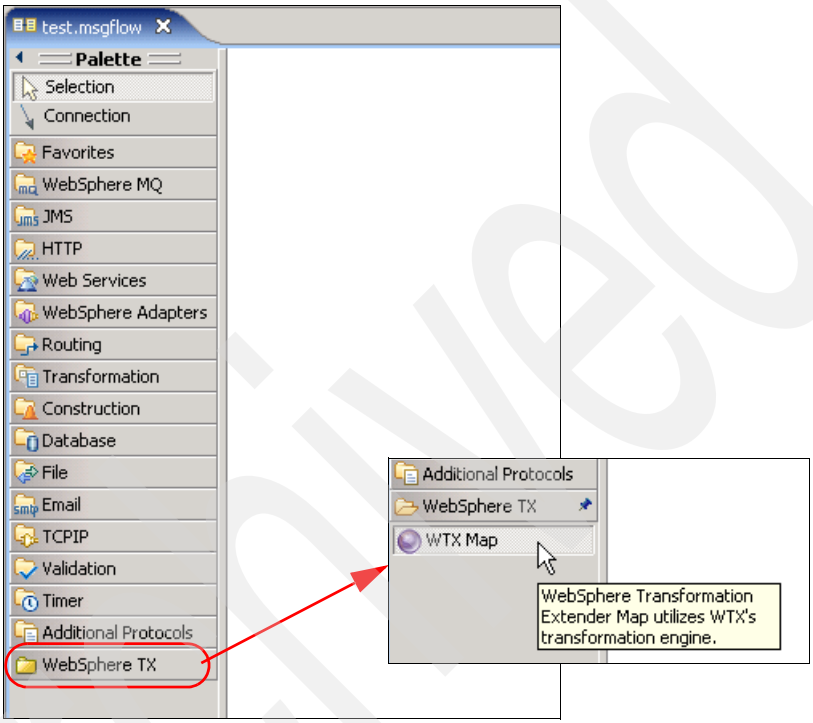

Figure 4-39 The Message Broker Palette

4.4.2 Using WebSphere Transformation Extender for Integration Servers with WebSphere Message Broker

After you successfully install the necessary components for WebSphere Transformation Extender for Message Broker, you can begin to use its capabilities.

Using and configuring the WTX Map node

With the WTX Map node, you can call a WebSphere Transformation Extender map from a WebSphere Message Broker flow. The WTX Map node can be placed in a message flow in the same way as any other WebSphere Message Broker node.

When you place the node on the message flow design canvas, it originally has one terminal on the input and a failure terminal on the output, as shown in Figure 4-40.

Figure 4-40 An unconfigured WTX Map node

You can then configure the node in the Properties window.

The Basic tab

In the Properties window, click the **Basic** tab (Figure 4-41) to define for the mapping node the WebSphere Transformation Extender map that it should call.

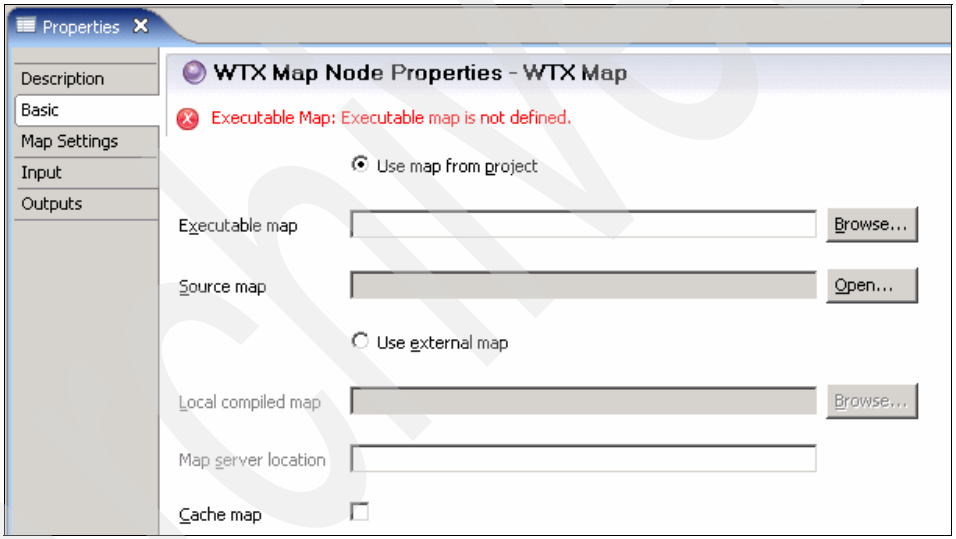

Figure 4-41 The WTX Node Basic tab

You have two options:

- Click the **Use map from project** radio button if you have the map source file (.mms) in your workspace. By clicking this option, you can simply point the node to an executable map within this map source. In this case, the map is compiled and added to the broker archive (BAR) file automatically when the flow is compiled. For more information, see “Deploying a WTX map to WebSphere Message Broker” on page 145.
- Click the **Use external map** radio button if you do not have the map source file in your workspace. In this case, you must fill in the Map Server Location

field with the fully qualified path and name of the compiled map on the WebSphere Message Broker server.

If you have a local copy of the compiled map on your development workstation (not necessarily in your workspace), enter the location of this compiled map file in the Local compiled map field. The output terminals are then automatically created for you. See also “Creating output terminals” on page 141.

External map: If you are using an external map, you *must* ensure that the compiled map file is on the broker runtime system at the location configured in the node.

The **Basic** tab also has a *Cache map* check box. If you select this check box, the map remains in memory after having been loaded for the first time. There are several reasons to do this:

- Map caching has a positive impact on performance.
- With map caching, you can run multiple instances of the same map in parallel. If you configure your message flow to run with additional instances (done in the BAR file editor), you must enable map caching.

The Map Settings tab

If you have a local copy of the map (either the map source in your workspace or the compiled map anywhere on your workstation), you can override map settings on the **Map Settings** tab. Click the **Map settings** button (Figure 4-42) to open the Map Settings editor.

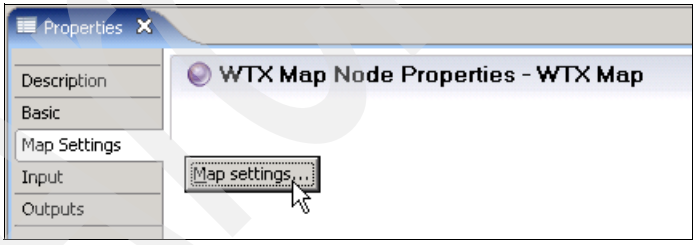

Figure 4-42 Opening the Map Setting editor

By using the Map Settings editor, you can override properties of a map. Typical settings that you override include Map Trace, Map Audit, and so on. Figure 4-43 shows the Map Settings editor.

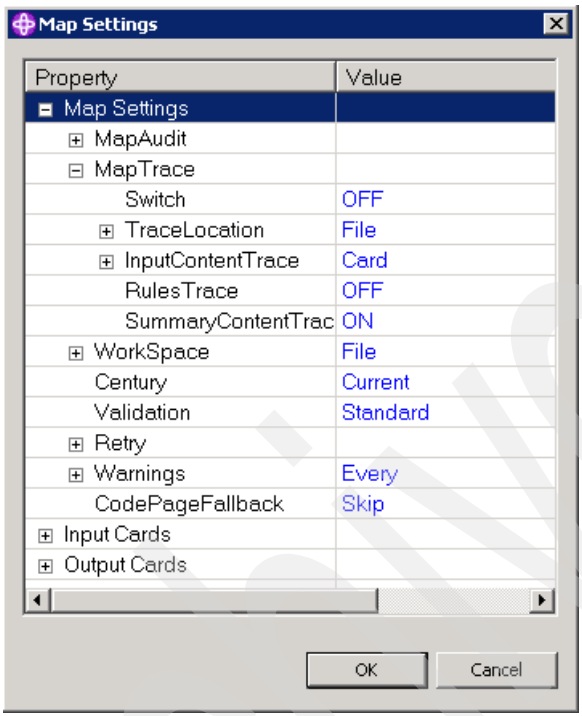

Figure 4-43 The Map Settings editor

Overriding the map settings: You can only override map settings for maps in which you have a local copy on your development workstation. Whether the Map settings button is active or disabled depends on the settings on the **Basic** tab:

- The button is *active* if you select either the Use map from project option or Use external map option with a Local Compiled map.
- The Map Settings button is *disabled* if you select the Use external map option and enter only the map server location.

The Input tab

Use the **Input** tab (Figure 4-44) to configure which of the map's input cards that you want to feed from the message flow. The Card number to wire field defaults to 1. You can change this value to any other input card that you want to use.

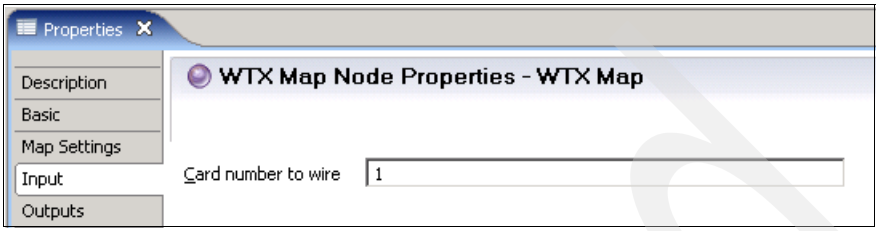

Figure 4-44 The Input tab

If you use the WTX Map node without a Collector node, only the input card that you specify receives its input data from the WebSphere Message Broker. All other input cards use their normal WebSphere Transformation Extender adapters (configured on the cards) to read the data. For more information about how to feed more than one input card from the message flow, see "Feeding multiple input cards from WebSphere Message Broker" on page 142.

The Outputs tab

On the **Outputs** tab (Figure 4-45), you specify which output cards you want to feed back to the message flow and how they should be represented. You can configure any number of output cards to be fed back into the message flow. Output cards that are not fed back into the message flow use their normal WebSphere Transformation Extender adapter that you configured when you developed the map.

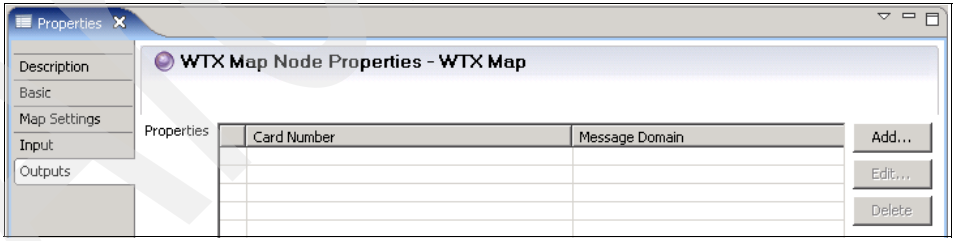

Figure 4-45 The Outputs tab

Click the **Add** button to configure an output card. In the Add Properties entry window (Figure 4-46 on page 140), you can specify the following items:

- The output card number
- Message Domain

  This is the domain in which you want the data to be parsed after it enters the message flow. It can be a BLOB, MRM, XMLNSC, or another Message Broker domain.
- Message Set

  You provide this information if you use a Message Set (XMLNSC or MRM) in WebSphere Message Broker.
- Message Type

  You enter this type if you use an MRM Message Set in WebSphere Message Broker.
- Message Format

  You specify this format if you use an MRM Message Set in WebSphere Message Broker.
- Coded Character Set
- Encoding

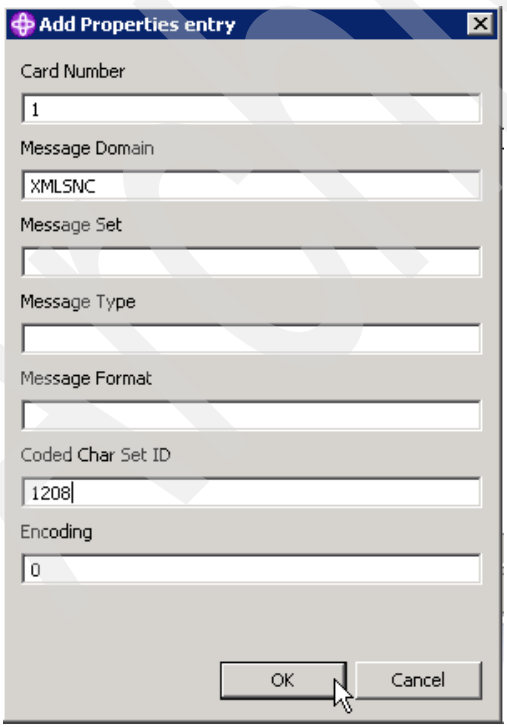

Figure 4-46 Adding the properties

Enter the desired properties for every output card that you want to route back into the message flow. Every output card that you configured is shown on the **Outputs** tab (Figure 4-47).

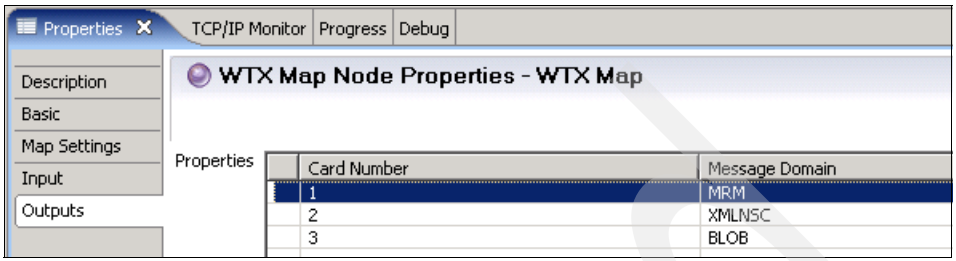

Figure 4-47 Output cards configured

Creating output terminals

If you want to wire output cards back into the message flow, you need output terminals on the node, as shown in Figure 4-48.

Figure 4-48 A WTX Map node with output terminals

There are two ways to create these output terminals:

- If you configured the node to point to a map (source or compiled) that is available on your system (see also “The Basic tab” on page 136), the output terminals are created automatically for you.
- If you refer to a map on the runtime server and you do not have a local copy on your machine, the node cannot know how many output cards the map has. Therefore, you must create the output terminals manually by right-clicking the node and selecting **Add Output Terminal** (Figure 4-49).

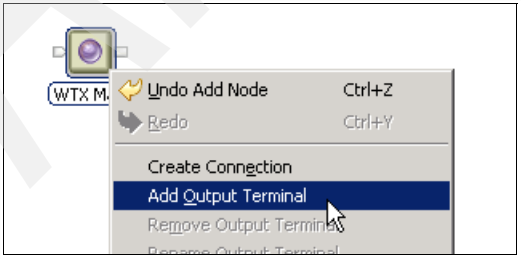

Figure 4-49 Adding an output terminal manually

As shown in Figure 4-50, you can then enter the name of the terminal and click **OK**.

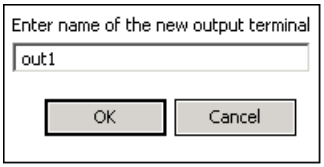

Figure 4-50 Entering the name of the output terminal

Output terminal names: Use only the names out1, out2, out3, ..., and so on for the output terminals. The rule is that the output terminals *must* be named outX, where X is the corresponding WebSphere Transformation Extender output card number.

Feeding multiple input cards from WebSphere Message Broker

It is possible to feed more than one input card with data coming from the message flow. You place a Collector node in front of the WTX Map node, as shown in Figure 4-51.

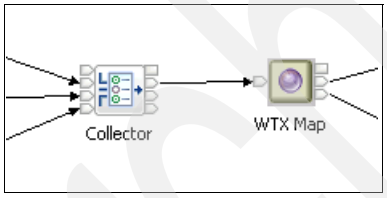

Figure 4-51 Feeding multiple input cards by using a Collector node

The Collector node is a general WebSphere Message Broker node. It is not specific to WebSphere Transformation Extender for Integration Servers, but it works well in combination with the WTX Map node. The node can collect data from one or more sources, and based upon configured criteria, join them in a message collection. A *message collection* is a grouping of the different input data structures for further processing by nodes downstream in the flow.

Figure 4-52 shows a debugger view of a message structure that is propagated by a collector node. You can distinguish two different data elements, both using the XMLNSC domain.

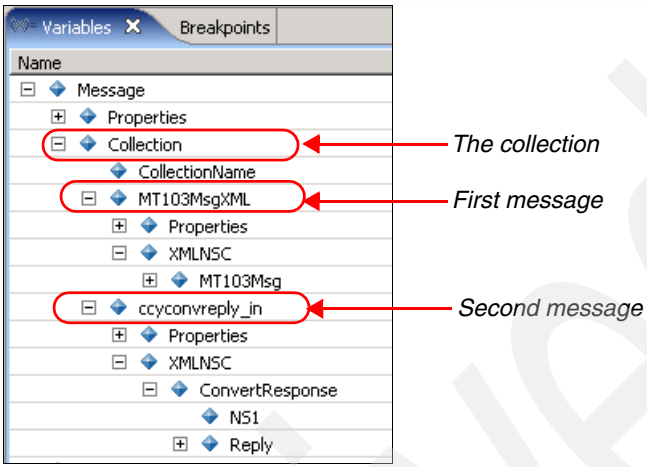

Figure 4-52 A collection with two messages

A Collector node used in conjunction with a WTX Map node has a special feature. That is, it can pass the data of multiple input sources straight to the different input cards of a map. To configure this capability:

1. Place the Collector node in front of the WTX Map node.
2. Equip the node with input terminals. Right-click the **Collector node** and select **Add Input Terminal** (Figure 4-53).

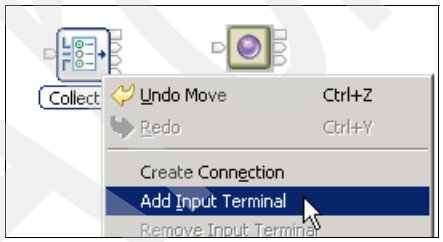

Figure 4-53 Adding an input terminal for the Collector node

3. In the window that opens (Figure 4-54), enter the name of the input terminal. The name of the terminal *must* match the name of the input card into which you want to pass the data.

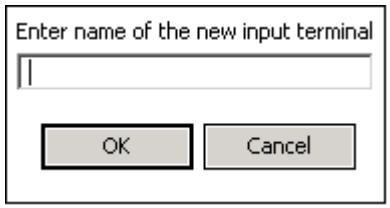

Figure 4-54 Naming the input terminal

Important: Name your input terminals the same as the input cards to which you want them to connect. The rule is that the names of the input terminals of the Collector node *must match* the names of the corresponding WebSphere Transformation Extender input cards of the map called in the WTX Map node.

4. Repeat steps 2 and 3 for any other input terminals that you must create.
5. On the **Basic** tab of the Properties view (Figure 4-55 on page 144), enter a Collection expiry (in seconds). The *collection expiry* is the time the node will wait for a collection to complete. If the collection is not completed by that time, the incomplete collection is sent out through the Expire terminal.

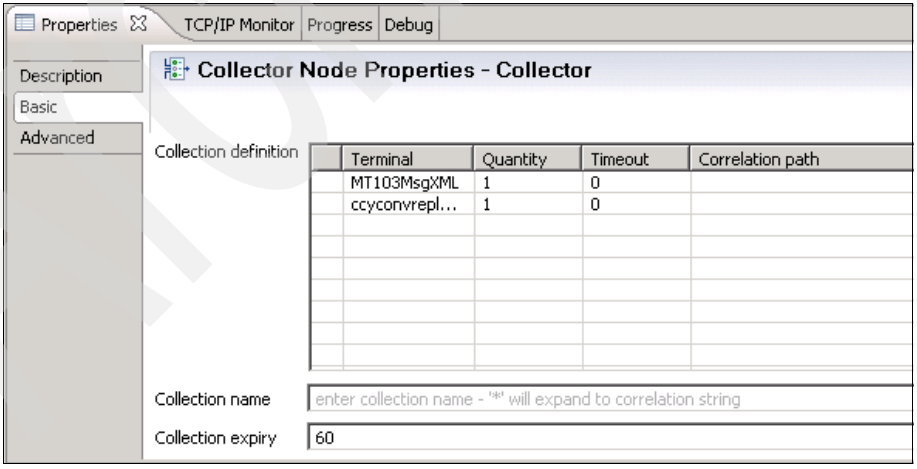

Figure 4-55 Collector Node Properties on the Basic tab

Wiring terminals: You can wire both the Out terminal and the Expire terminal of the Collector node to the In terminal of the WTX Map node. If you wire these terminals and a collection is incomplete, a map return code of 8 (`One or more inputs is invalid`) is triggered.

6. Wire the Catch terminal of the Collector node to one or more nodes that will perform the error handling if an exception occurs downstream of the Collector node. If you do not connect the Catch terminal of the Collector node and such an exception occurs, the Collector node continuously retries to process the collection.

For more information about the Collector node and its settings, see the “Collector node” article of the WebSphere Message Broker Information Center at the following address:

`http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r1m0/index.jsp?topic=/com.ibm.etools.mft.doc/ac37820_.htm`

Deploying a WTX map to WebSphere Message Broker

The way in which a map is deployed on the runtime server depends on the configuration that you made on the **Basic** tab of the WTX Map node.

Deploying maps configured with available sources

If the map source is available on your development machine and you selected the Use map from project option (Figure 4-56), the compilation and deployment of the map is fully integrated with the compilation and deployment of the message flow that uses it.

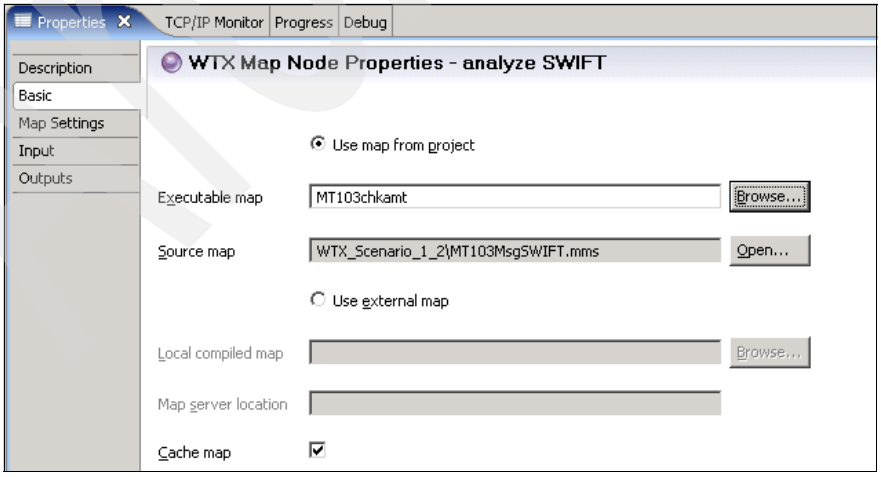

Figure 4-56 Using a map from a local project

At the time at which you create a BAR file that contain the artifacts to deploy, the Message Broker Toolkit generates a map archive (MAR) file and adds it to the BAR file as shown in Figure 4-57. A MAR file contains the compiled versions of one map for Windows, UNIX, and z/OS systems. Therefore, it can be deployed to any WebSphere Message Broker environment. Figure 4-57 shows an overview of how both files relate to each other.

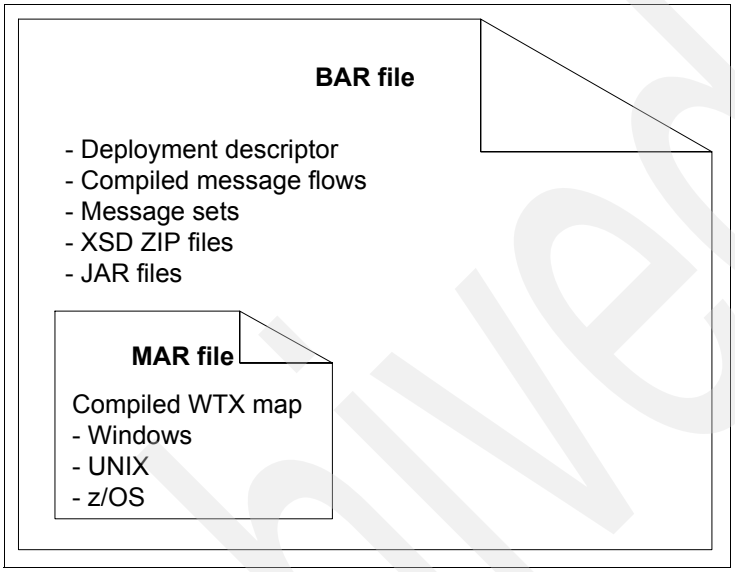

Figure 4-57 The BAR and MAR files

You can use the Deployment Descriptor in the BAR file to override certain properties of the WTX Map node. Figure 4-58 on page 147 shows a BAR file that contains several flows. One of these flows has two WTX Map nodes. Therefore, the tooling automatically generated two MAR files and added them to this BAR file.

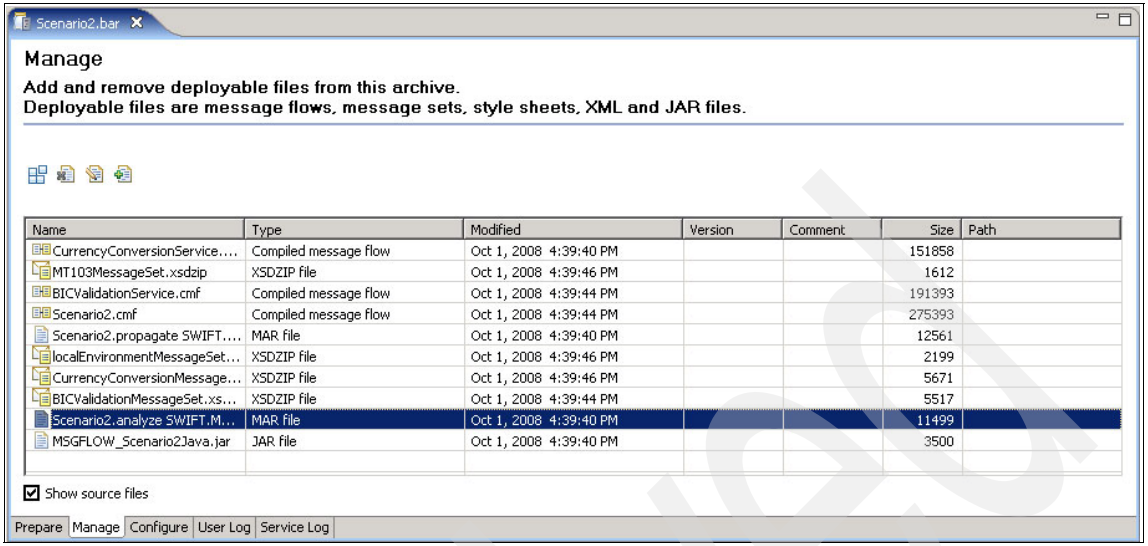

Figure 4-58 A BAR file containing MAR files

Figure 4-59 shows the **Configure** tab of the BAR file. You can navigate to the WTX Map nodes and override selected properties of the WTX Map nodes before deploying the flows and maps to the runtime broker.

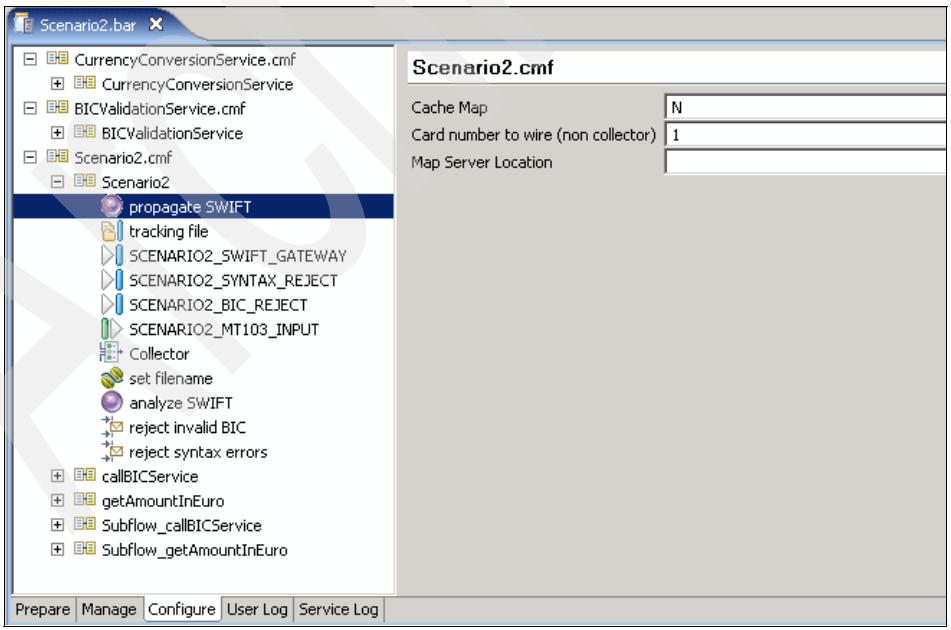

Figure 4-59 Overriding the WTX Map node properties

Deploying maps directly on the server

If you do not have the map source on your development machine and have selected that you want to refer to an external map, you *must* ensure that the compiled map exists at that specific location on the runtime server. In this case, the tooling does not do anything to automate the deployment.

Overriding the map location in the LocalEnvironment

You can dynamically override certain settings at run time by using the LocalEnvironment. To override the settings, you enter the following equally named values:

- LocalEnvironment.WTX.MapName
- LocalEnvironment.WTX.MapServerLocation
- LocalEnvironment.WTX.InputCardNumberToWire

For more information about the LocalEnvironment tree structure, see the “Local environment tree structure” article in the WebSphere Message Broker Information Center at the following address:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r1m0/index.jsp?topic=/com.ibm.etools.mft.doc/ac00520_.htm

Using the Resource Registry

You can use the WebSphere Transformation Extender Resource Registry in conjunction with WebSphere Message Broker. By using the Resource Registry, you can map resource aliases (assigned during development of the map) to concrete resources for the run time.

You must assign a resource registry file for each execution group with which you want to use the Resource Registry. Enter the **mqsisetproperties** command in WebSphere Message Broker. You can set or change each of the properties that are shown after the command.

```
mqsichangeproperties brokername
-e executiongroup
-o ComIbmWTXManager
-n resourceConfigFile
-v locaton_of_your_resource_registry_configuration_file
```

For more information about the Resource Registry, see 5.3.12, “Resource Registry” on page 332.

Type trees versus message sets

WebSphere Transformation Extender offers transparency in the use of type trees and message sets. If you have existing WebSphere Message Broker message sets, you can use them directly in a WebSphere Transformation Extender map, without needing to create additional type trees.

To use an existing message set:

1. Create a new card.
2. In the Project Browser window (Figure 4-60), in the Type Tree field, select the message set MXSD file.

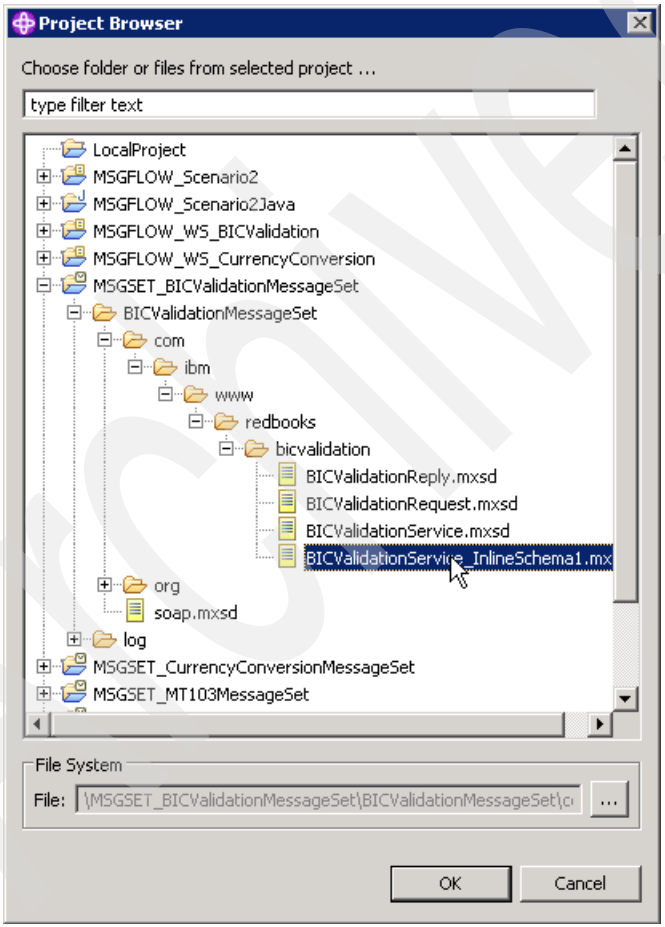

Figure 4-60 Selecting a WebSphere Message Broker MXSD file to use in a WebSphere Transformation Extender map

3. In the Select Type window (Figure 4-61), navigate to the group that represents the top-level element of what you want to map.

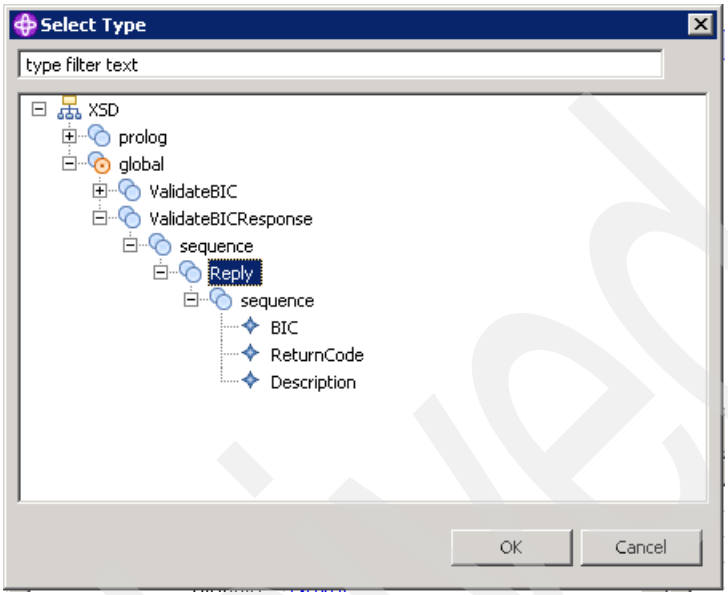

Figure 4-61 Selecting the type

4. Map this structure in the map editor as though you were using a classical type tree. Figure 4-62 shows a detail of the map editor using the Message Set structure.

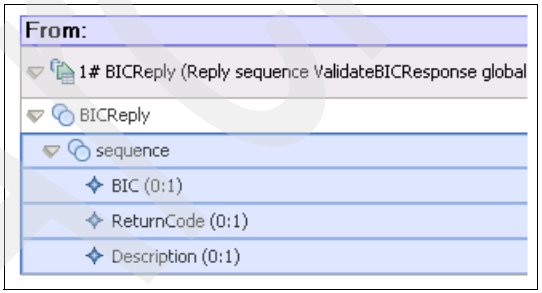

Figure 4-62 The message set in the map editor

Consulting the documentation for the WTX Map node

To view the documentation for the WTX Map node:

1. Install the Transformation Extender Online Library. You must do this after all the other WebSphere Transformation Extender installations.
2. Click **Start** → **All Programs** → **IBM WebSphere Transformation Extender 8.2** → **Transformation Extender Online Library** to start the Online Library.
3. Scroll down to the link **WebSphere Transformation Extender for Message Broker** and click the link to open the PDF documentation (Figure 4-63).

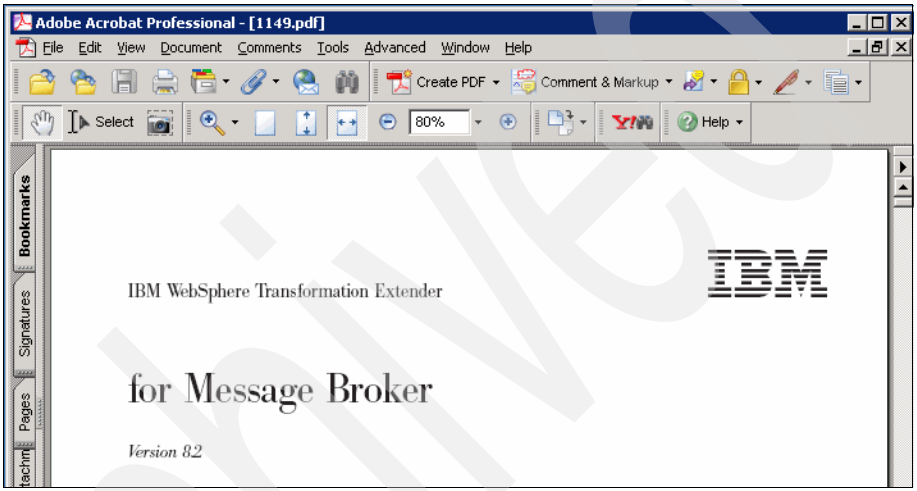

Figure 4-63 PDF documentation for WebSphere Transformation Extender for Message Broker

4.5 WebSphere Transformation Extender for Integration Servers with WebSphere ESB and WebSphere Process Server

WebSphere Transformation Extender for WebSphere Enterprise Service Bus and WebSphere Process Server are a subset of WebSphere Transformation Extender for Integration Servers, as shown in Figure 4-64.

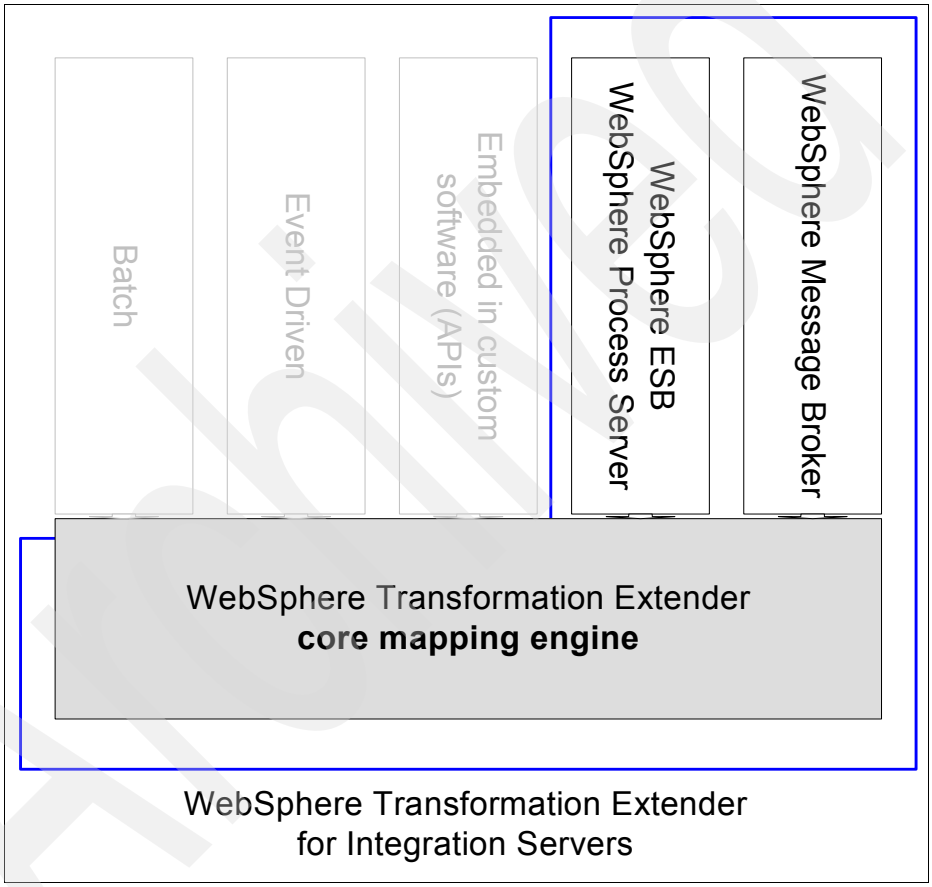

Figure 4-64 WebSphere Transformation Extender for Integration Servers

4.5.1 Installing WebSphere Transformation Extender for Integration Servers with WebSphere ESB and WebSphere Process Server

Figure 4-65 shows the software packages that you typically install to use WebSphere Transformation Extender with WebSphere ESB or WebSphere Process Server.

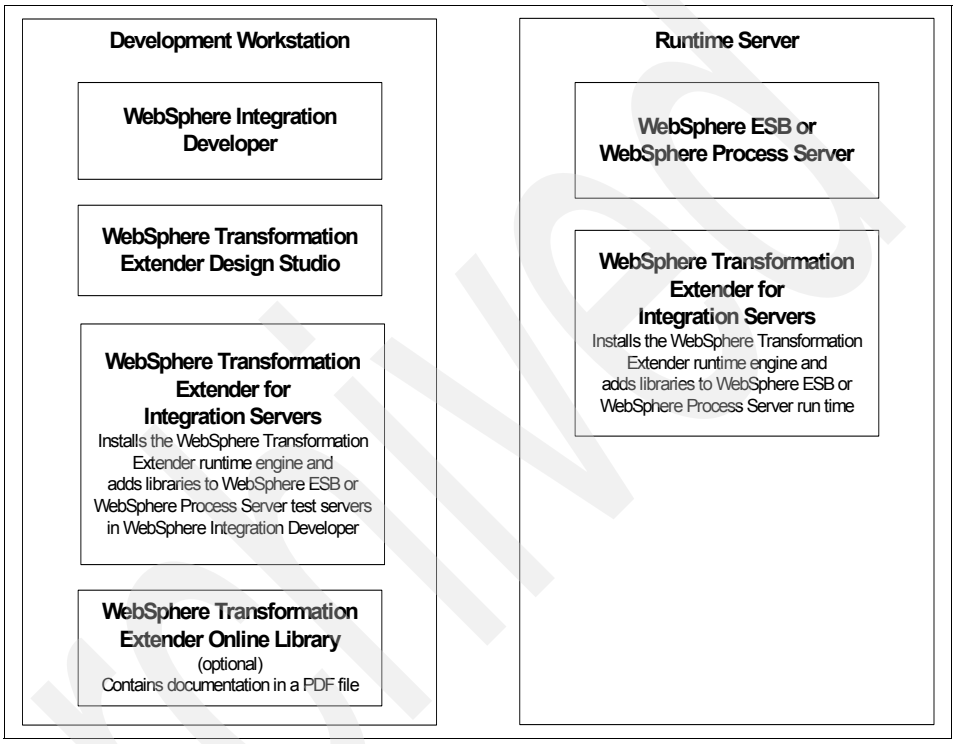

Figure 4-65 WebSphere Transformation Extender for Integration Servers with WebSphere ESB or WebSphere Process Server

To use WebSphere Transformation Extender maps in a WebSphere ESB or WebSphere Process Server environment, you must have WebSphere Transformation Extender for Integration Servers installed. The installation of WebSphere Transformation Extender must be performed *after* the installation of WebSphere ESB or WebSphere Process Server.

If you want to develop and compile maps for WebSphere ESB or WebSphere Process Server, you must also install the WebSphere Transformation Extender Design Studio on your development workstation. The WebSphere Transformation Extender Design Studio and WebSphere Integration Developer (the development tool for WebSphere ESB and WebSphere Process Server) can share the same

Eclipse platform. If you install the WebSphere Transformation Extender Design Studio *and* WebSphere Transformation Extender for Integration Servers after you install WebSphere Integration Developer, the Transformation Extender Development perspective is added to the list of available perspectives as shown in Figure 4-66.

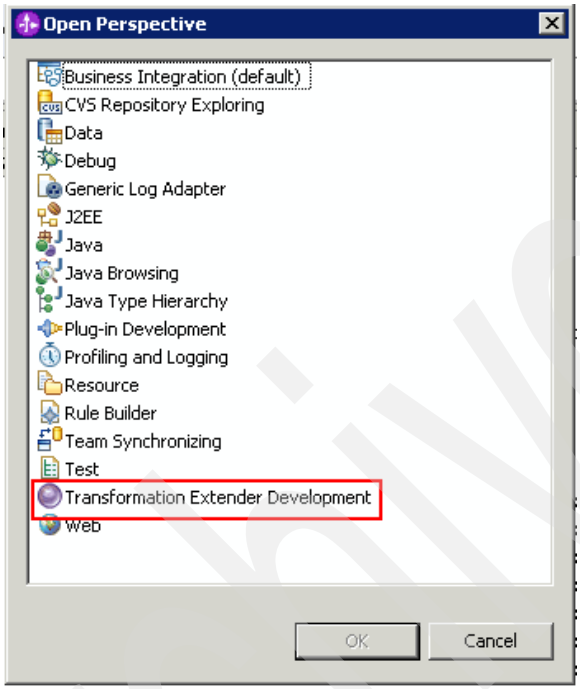

Figure 4-66 The Transformation Extender Development perspective in WebSphere Integration Developer

Note: To add the Transformation Extender Development perspective to other Eclipse installations or if the Transformation Extender Development perspective is not accessible in your WebSphere Integration Developer:

1. Go to the `<WTXDesignStudio>\wtx_eclipse\eclipse\links` folder.
2. Copy the `wtx_es.link` and `wtx_esdoc.link` files.
3. Paste the files to the `links` folder of your WebSphere Integration Developer or other Eclipse installation (Figure 4-67 on page 155).

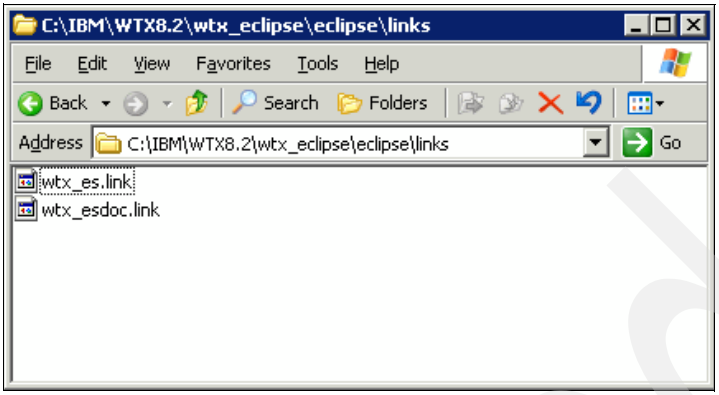

Figure 4-67 The location of WTX Eclipse plug-in link files

4.5.2 Using WebSphere Transformation Extender for Integration Servers with WebSphere ESB and WebSphere Process Server

In the previous section, we discussed the software packages and the approach that you use to typically install and use WebSphere Transformation Extender with WebSphere ESB or WebSphere Process Server. We now discuss the most beneficial ways in which you can use WebSphere Transformation Extender to transform the data between the specific data format in source applications and the XML-based objects that are used by WebSphere ESB or WebSphere Process Server. First we introduce the basic concept and value proposition for integration, and then we discuss several common applications.

Basic concept of the integration between WebSphere Transformation Extender and WebSphere ESB/WebSphere Process Server

WebSphere Transformation Extender complements the WebSphere Process Server or WebSphere ESB Service Component Architecture (SCA). All the data in WebSphere ESB or WebSphere Process Server is represented as XML-based objects. However, chances are that not all applications or services we want to integrate with use XML data. If we want to integrate WebSphere ESB or WebSphere Process Server with applications that use non-XML data formats, such as COBOL copybooks, tag delimited data, or complex structures used in industry standards (EDI, SWIFTNet, ACCORD, and so on), we must transform the data between the specific data format and the XML-based objects. This is a task for which WebSphere Transformation Extender is extremely well fitted.

The integration of WebSphere Transformation Extender and WebSphere ESB or WebSphere Process Server is based on the concept of *data bindings* and *data handlers*.

Using WebSphere Transformation Extender with WebSphere ESB or WebSphere Process Server and native data bindings

A *data binding* is a particular Java class that is used by WebSphere ESB and WebSphere Process Server components. Its task is to get the physical data and transform it to the internally used XML-based format and vice versa. The physical data can be messages on an MQ or Java Message Service (JMS) queue, or HTTP Web service calls.

If you must interact with data on MQ, JMS, or HTTP, and the data is not XML-based, you can use a special data binding class called the *WTX data binding*. The WTX data binding class executes maps that transform data from any native format to XML and vice versa. If the WTX data binding is assigned to an export component, then a NativeToXML map is called. If the WTX data binding class is assigned to an import component, an XMLToNative map is called.

The WTX data binding determines the map that should be called based upon the properties saved in a file called the *data binding configuration*. Figure 4-68 shows how the WTX data binding can use a WebSphere Transformation Extender map to convert the data.

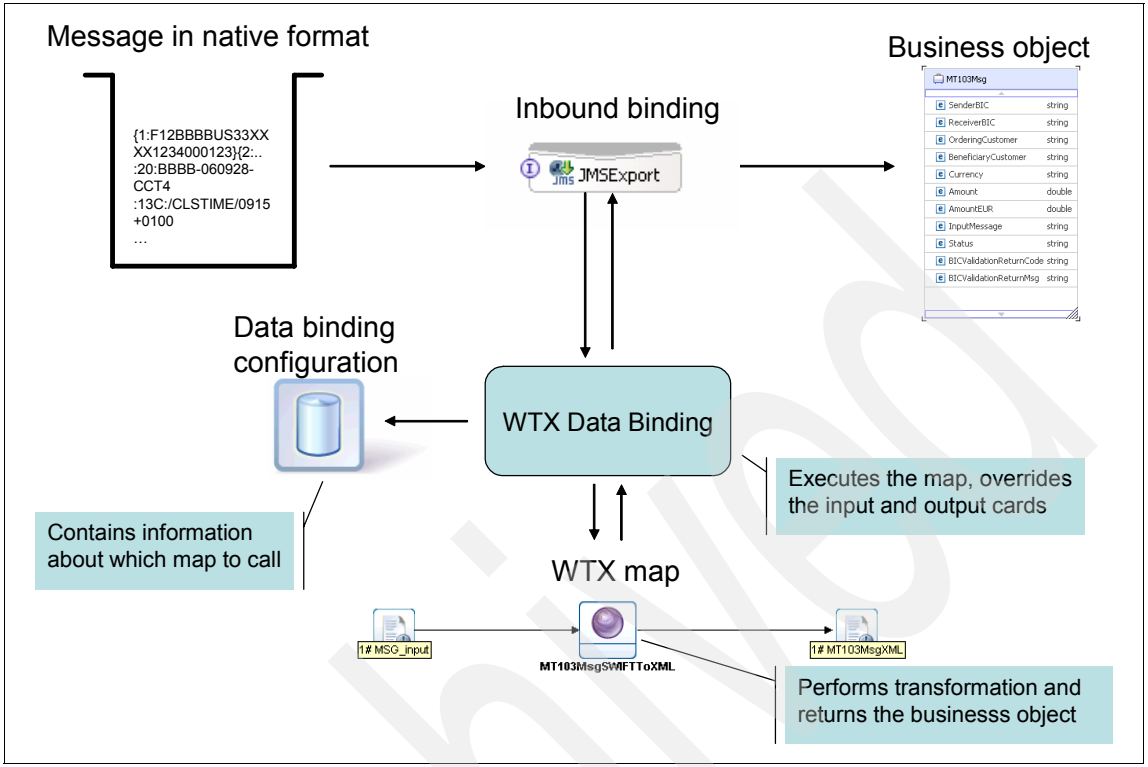

Figure 4-68 Using a WTX data binding

Using WebSphere Transformation Extender with WebSphere ESB or WebSphere Process Server and WebSphere Adapters

WebSphere ESB and WebSphere Process Server can also use WebSphere Adapters. These technology adapters are extensions to the native data bindings and can read and write data from flat files, FTP, and Email.

The WebSphere Adapters have their own specific data binding class to handle the specifics for the application or technology to which they connect. For the transformations, they can use a *data handler* class.

WebSphere Transformation Extender provides a *WTX map selection data handler* class that can be used by these adapter-specific data binding classes. The adapter-specific data binding classes delegate the transformations between the native format and the XML-based data object to WebSphere Transformation Extender.

Figure 4-69 shows a Flat File adapter binding that uses the WTX data handler to call a WebSphere Transformation Extender map.

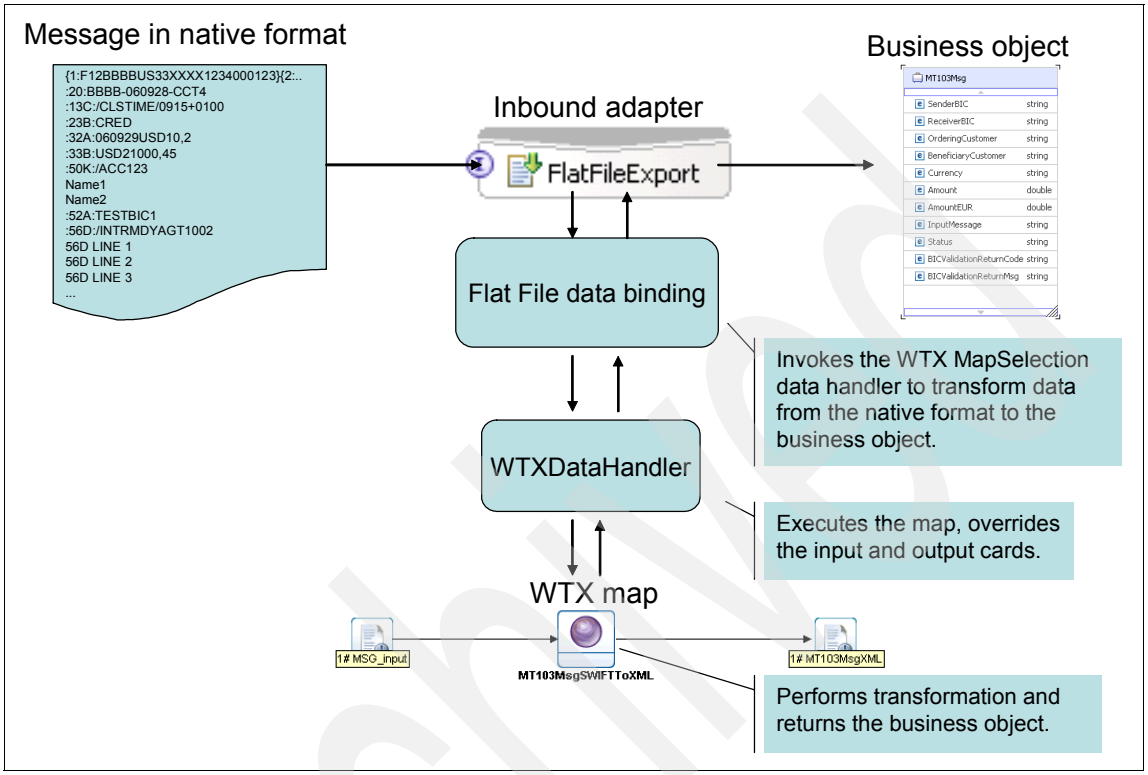

Figure 4-69 Example of using the WTX Map Selection data handler with a Flat File adapter

Developing maps for WebSphere ESB or WebSphere Process Server

Maps for WebSphere ESB or WebSphere Process Server should always have one input card and one output card. The NativeToXML map has one output card, which is based on the XML schema of the desired business object, and one input card, which describes the native data type.

The WebSphere Transformation Extender maps for WebSphere ESB or WebSphere Process Server are always constructed in the same way. The WebSphere Transformation Extender Design Studio contains a wizard called *Map Generator for WESB/WPS*. This wizard helps generate type trees and maps based on the business object schema.

The wizard creates the map source file. This map source file contains two maps. The names of the maps are created according to the following structure:

- The map from the XML to the native format is called *$BOName$NativeToXML*.
- The map from the native format to XML is called *$BONameXMLTo$Native*.

In these names, *$BOName* is a name of the business object, and *$Native* is a name of the native format. The Map Generator Wizard requires you to provide both the names of the business object and the native format. One card (respectively input or output) is automatically added by the wizard. The type used in those cards is a built based on a business object schema.

Note: The business object schema cannot be used directly as a type tree in the Map Editor. The Map Generator for WebSphere ESB or WebSphere Process Server (Figure 4-70) creates a new XSD schema file that refers to the business object schema and contains an element of its type.

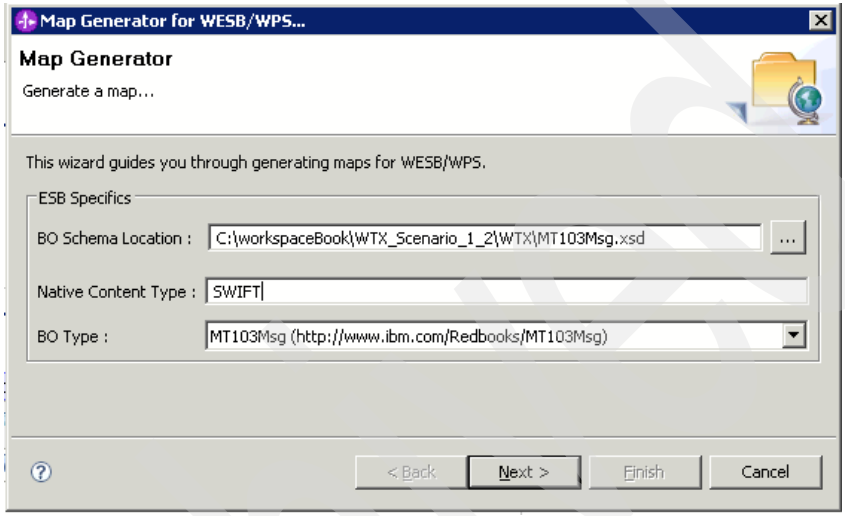

Figure 4-70 Map Generator for WESB/WPS

Figure 4-71 on page 161 and Figure 4-72 on page 161 illustrate the difference between a business object schema generated by WebSphere Integration Developer and a schema that can be used in WebSphere Transformation Extender. See also 8.3, “Building the WebSphere ESB scenario” on page 490.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.ibm.com/Redbooks">
   <xsd:complexType name="Type1">
      <xsd:sequence>
         <xsd:element minOccurs="0" name="field1"
            type="xsd:string">
         </xsd:element>
         <xsd:element minOccurs="0" name="field2"
            type="xsd:string">
         </xsd:element>
         <xsd:element minOccurs="0" name="field3"
            type="xsd:string">
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>
</xsd:schema>
```

Figure 4-71 A business object schema as generated by WebSphere Integration Developer

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.ibm.com/Redbooks">
   <xsd:complexType name="Type1">
      <xsd:sequence>
         <xsd:element minOccurs="0" name="field1"
            type="xsd:string">
         </xsd:element>
         <xsd:element minOccurs="0" name="field2"
            type="xsd:string">
         </xsd:element>
         <xsd:element minOccurs="0" name="field3"
            type="xsd:string">
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>
<xsd:element name="Element1" type="Type1"/>
</xsd:schema>
```

Figure 4-72 A schema suitable for use with WebSphere Transformation Extender

It is up to the developer to create the type tree for the native side of the map and to build the transformation logic between the business object and the native format. Then the maps must be compiled, and the compiled maps must be moved to the WebSphere ESB or WebSphere Process Server project. The actual runtime environment does not need the map source file.

Compiling maps: Remember to compile the map for the platform on which your WebSphere ESB or WebSphere Process Server server is running. If you are not sure on which platform your maps will be executed, compile the maps for all platforms to make your project fully platform independent.

All the artifacts created by the Map Generator for WESB/WPS are stored in the WTX folder. (This folder is automatically created if it does not exist.) By default, the WTX data binding looks for compiled maps in the WTX folder. This default location can be changed in the WTX data binding configuration file.

For an example of how to use the WTX Map Generator for WESB/WPS, see the integration scenario in 8.3, “Building the WebSphere ESB scenario” on page 490.

Using the WTX data binding and the WTX data handler

To use WebSphere Transformation Extender maps in your WebSphere Process Server or WebSphere ESB project, you must copy the compiled maps from the WebSphere Transformation Extender project to the WebSphere ESB or WebSphere Process Server module or a referred library. Then you can configure the WTX data binding or WTX data handler.

The WTX data binding can be used with JMS, generic JMS, MQ-JMS, MQ, and HTTP exports and imports.

The WTX Map Selection data handler can be used in an enterprise information system (EIS; adapter) context for FlatFile, FTP, and Email imports and exports.

To configure an import or export component to use a WTX data binding, you must create a *binding resource configuration* file. The same applies to the use of a WTX data handler. A binding resource configuration file is created by the New Binding Resource Configuration wizard. An example of using this wizard is

shown in the integration scenario in 8.3, “Building the WebSphere ESB scenario” on page 490. Figure 4-73 shows a configuration file that has been generated by the wizard.

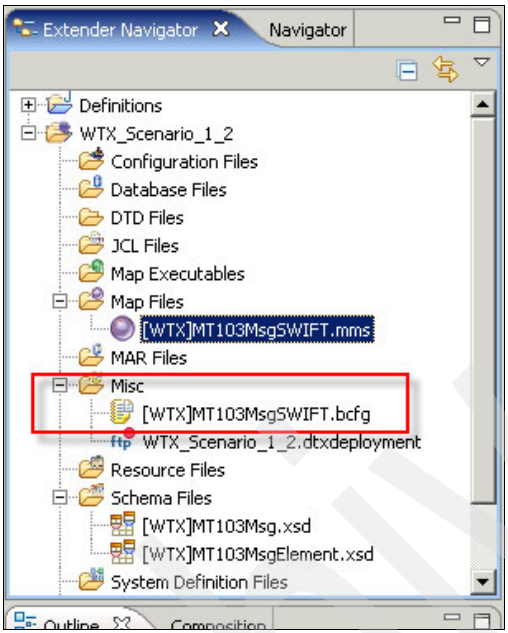

Figure 4-73 The Data binding resource configuration file created by Map Generator for WESB/WPS in the Transformation Extender Development perspective

Copy the .bcfg file from your WebSphere Transformation Extender project to your WebSphere ESB or WebSphere Process Server module. It is displayed in the Binding Resources folder in the Business Integration view, as shown in Figure 4-74 on page 164.

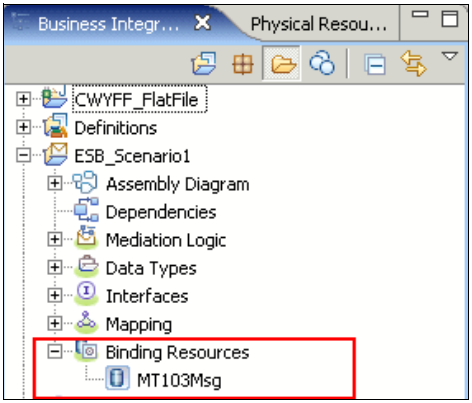

Figure 4-74 The binding resource configuration file in the Business Integration perspective

The binding resource configuration file for WTX data binding contains the following properties (Figure 4-75):

- Business Object
- Content Type
- TransformToNative Map Name
- TransformToXML Map Name

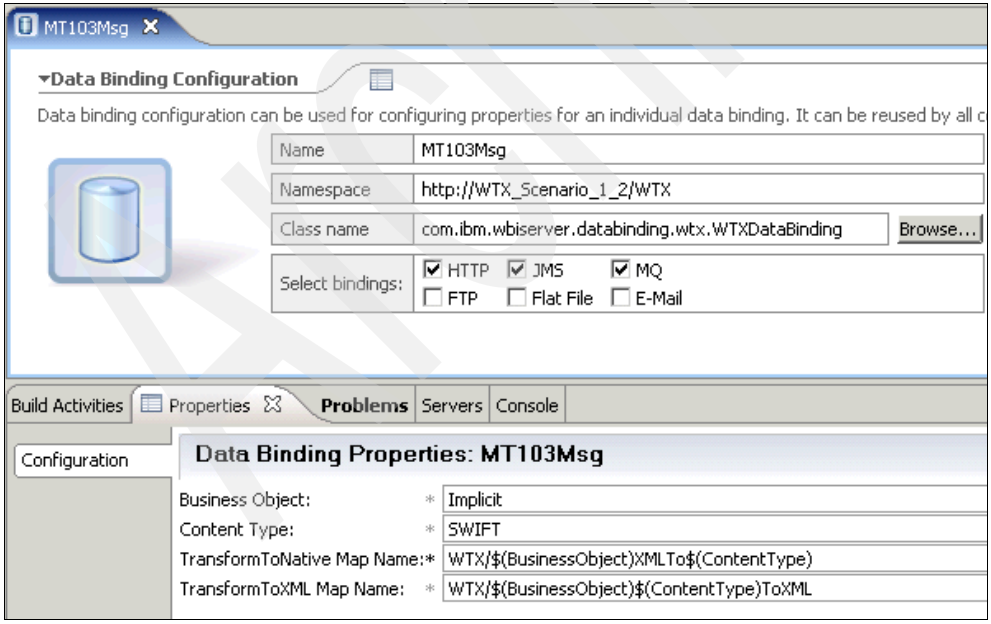

Figure 4-75 The WTX data binding resource configuration

By default, the value of the property Business Object is set to **Implicit**. This means that the value of the property depends on the operation parameter of the service interface. For example, if operation = create (MT103Msg MT103MsgBO), then the property value = MT103Msg. See 8.3, “Building the WebSphere ESB scenario” on page 490, for an example.

The names of the maps are built from a regular expression and resolved at run time. By default, the WTX folder is suggested, but you can change it. You can also hardcode the map name. However, we do not recommend this practice.

Extension for compiled maps: Do *not* use the extension for compiled map names. The appropriate extension for the specific platform is chosen at run time.

4.6 WebSphere Transformation Extender for Application Programming edition APIs

With the WebSphere Transformation Extender for Application Programming edition, you can embed WebSphere Transformation Extender maps in your own software (Figure 4-76 on page 166). You get a set of APIs that you can use to call out to the core mapping engine.

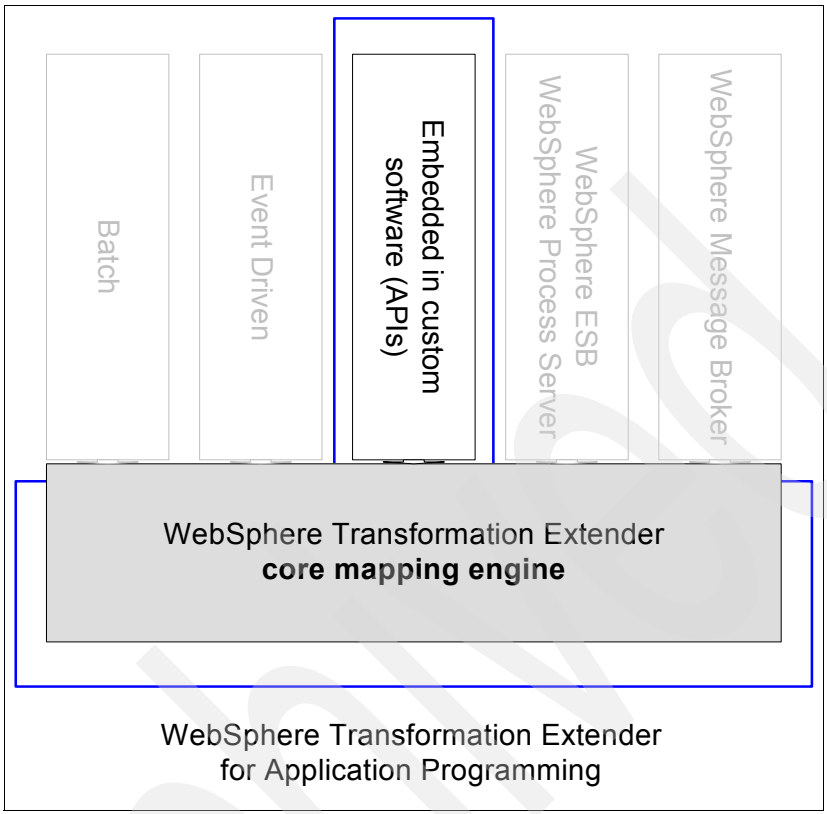

Figure 4-76 WebSphere Transformation Extender for Application Programming

4.6.1 Installing WebSphere Transformation Extender for Application Programming edition APIs

Like all other editions, the installation of WebSphere Transformation Extender for Application Programming is straightforward. On a Windows machine, you must run the `setup.exe` file and click through the pages of the installation wizard.

On development machines, you must also install the WebSphere Transformation Extender Software Developer Kit (SDK). The SDK gives you the API libraries and a set of examples for each API. By using this example code, you can easily learn how to use the APIs.

If you want to use the Online Library to access information about the APIs, install it after the other WebSphere Transformation Extender software. The installer detects that the WebSphere Transformation Extender for Application Programming edition APIs is installed and adds the PDFs for the APIs.

Figure 4-77 shows the software packages that you install on a development server and a runtime server.

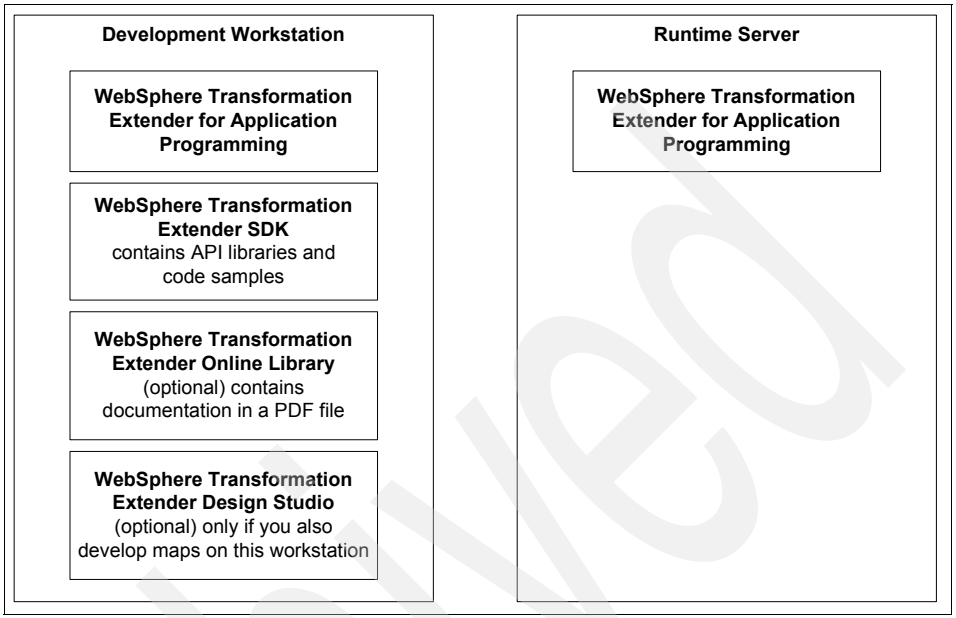

Figure 4-77 Packages to install in order to use the APIs

4.6.2 Using WebSphere Transformation Extender for Application Programming edition APIs

Several APIs are available for WebSphere Transformation Extender. While we explain the different APIs in depth, we refer you to the location where you can find sample code. To get the sample code, install the WebSphere Transformation Extender SDK. In the descriptions that follow, we assume that you have installed the SDK in the `C:\IBM\WTX8.2` directory.

Java API

With the Java API, you can embed WebSphere Transformation Extender maps in a Java program. By using this API, you can execute the map, override input and output cards with data streams, and so on. You can find the samples in the `C:\IBM\WTX8.2\examples\dk\dtxpi\java` directory.

RMI API

The Remote Method Invocation (RMI) API is much like the Java API. The difference is that the maps are executed on a central server that runs an RMI server. For the samples, navigate to the `C:\IBM\WTX8.2\examples\dk\dtxpi\rmi` directory.

EJB API

With the Enterprise JavaBeans (EJB) API, you can invoke WebSphere Transformation Extender maps as EJB session beans. For the samples, navigate to the `C:\IBM\WTX8.2\examples\dk\dtxpi\EJB` directory.

C/C++ API

By using the C/C++ API, you can embed WebSphere Transformation Extender in your own C/C++ programs. You can find the C code samples in the `C:\IBM\WTX8.2\examples\dk\dtxpi\c` directory.

COM API

With the COM API, you can invoke maps by using the COM interface. For sample code, navigate to the `C:\IBM\WTX8.2\examples\dk\dtxpi\com` directory.

CORBA API

With the CORBA API, you can invoke a map by using a CORBA object. For sample code, navigate to the `C:\IBM\WTX8.2\examples\dk\dtxpi\corba` directory.

C# API

The C# API is shipped as “examples only.” Look in the `<WTX installation path>\examples\dk\dtxpi\csharp` path. The C# API is a wrapper on the COM API.

COBOL API on z/OS

The WebSphere Transformation Extender provides a way to call the Platform API from COBOL programs by calling the Platform API functions as a DLL. This requires an IBM Host COBOL compiler that provides COBOL DLL support. You can find the code samples in `PDS DTX.SDTXSAMP : DTXTCCOB`.

PL/I API on z/OS

The WebSphere Transformation Extender installation includes the DTXTPLI example program, in the `DTX.SDTXSAMP` partitioned data set (PDS). The IBM Enterprise PL/I for z/OS compiler was used to compile the DTXTPLI example program.

Part 3

Getting started: Implementation using WebSphere Transformation Extender

In this part, we show you how to get started with WebSphere Transformation Extender. We include details about both the installation and use of the WebSphere Transformation Extender Design Studio for development, executing the maps in the run time, and troubleshooting. We also provide specific coverage about using WebSphere Transformation Extender on System z.

This part includes the following chapters:

- Chapter 5, “Developing in Design Studio” on page 171
- Chapter 6, “WebSphere Transformation Extender on System z” on page 353
- Chapter 7, “Troubleshooting” on page 443

5

Developing in Design Studio

In this chapter, we review the installation of the WebSphere Transformation Extender Design Studio. Then we describe in detail how to build, test, and use a transformation by using the WebSphere Transformation Extender Design Studio.

5.1 Installing WebSphere Transformation Extender Design Studio

The WebSphere Transformation Extender Design Studio is where you design, develop, test, and deploy your transformation maps, trees, routing rules, and other basic integration executables across your server environment. The Design Studio is an Eclipse-based tool that contains different editors for manipulating trees and maps. In this chapter, we refer to the Type Tree Editor and Map Editor when we want to reference anything specific in one of the tools. Design Studio also comprises the Integration Flow Designer and Database Interface Designer, of which neither currently is Eclipse-based.

Design Studio is installed on and operates currently only on Windows platforms. Version 8.2.0.3 has the following requirements:

- Pentium® III-based processor minimum with an XVGA monitor and video card
- At least 1 GB of free hard-disk space
- 256 MB minimum of RAM (2 GB recommended)

Design Studio installs on Windows 2000, Windows 2003, Windows Vista®, or Windows XP Professional. To install Design Studio:

1. Run the Design Studio `Setup.exe` program.
2. In the Choose Setup Language dialog box (Figure 5-1), select your language and click **OK**.

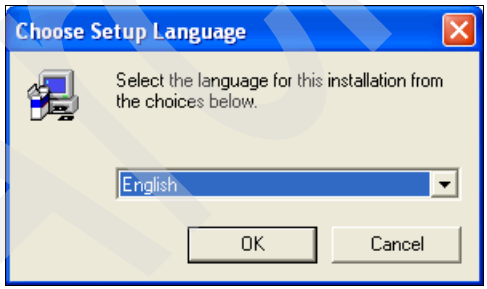

Figure 5-1 Choose Setup Language window

Note: In the Choose Setup Language dialog box, you can select the language of the installer. It does not affect the language of the WebSphere Transformation Extender Design Studio Eclipse GUI, which depends on the selected language of your regional settings.

3. When you see the Welcome window (Figure 5-2), ensure that you have temporarily disabled any anti-virus programs and have shut down all programs. Then click **Next**.

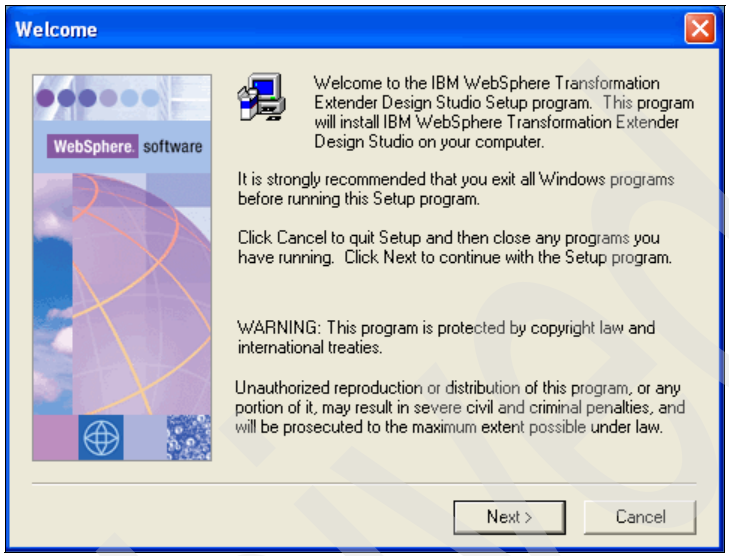

Figure 5-2 Welcome window

4. Review the Software License Agreement (Figure 5-3) and click **Yes**.

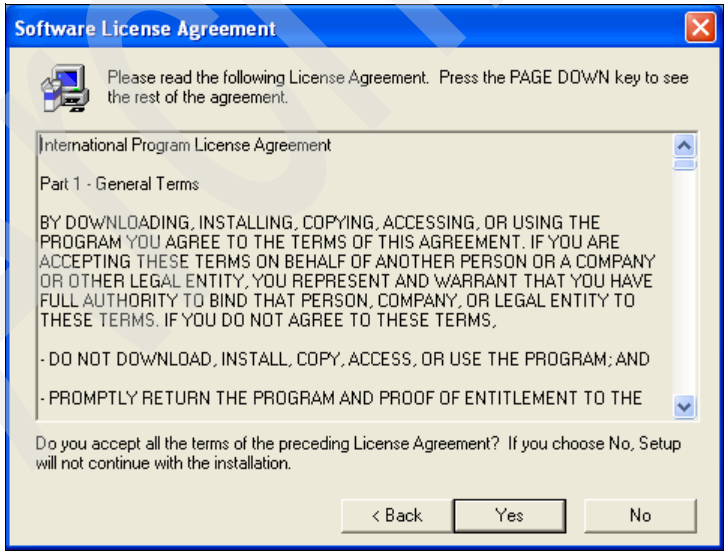

Figure 5-3 Software License Agreement window

5. In the User Information window (Figure 5-4), complete your name and company name and click **Next**.

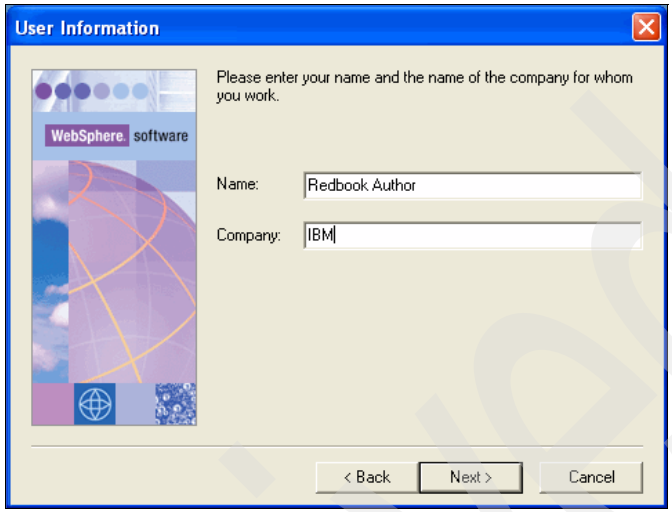

Figure 5-4 User Information window

6. In the Choose Destination Location window (Figure 5-5):
   a. Click **Browse** to set a shorter path, which is preferable. For example, something similar to the `C:\IBM\WTX8.2` path is recommended. The default installation path is `C:\IBM\WebSphere Transformation Extender 8.2`.

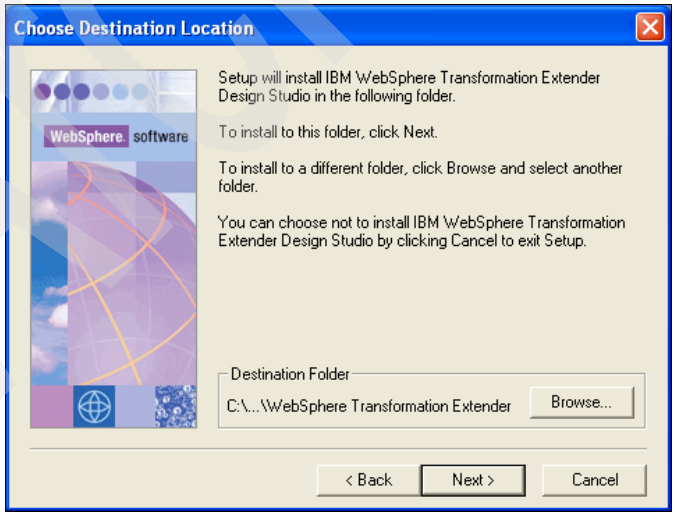

Figure 5-5 Choose Destination Location window

b. In the Choose Folder dialog box (Figure 5-6), either type a new path (or edit the existing path) or select the appropriate path to select the destination folder. Click **OK**.

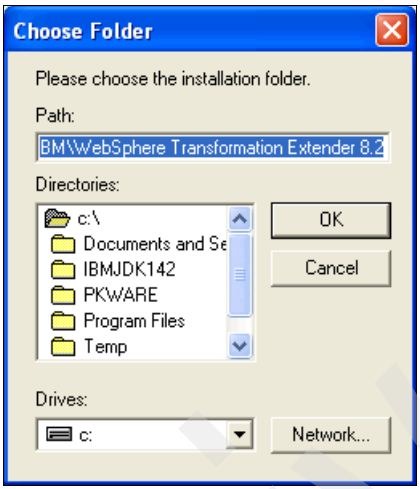

Figure 5-6 Choose Folder dialog box

c. If the selected folder does not exist, in the Setup dialog box (Figure 5-7), which validates the creation of the chosen folder, click **Yes** to continue if you have chosen the correct path. Click **No** if you want to go back and change your destination folder.

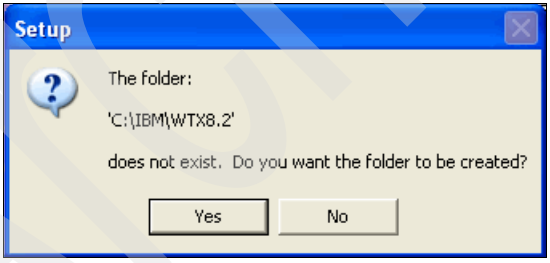

Figure 5-7 Setup dialog box to create new folder

d. Back in the Choose Destination Location window (Figure 5-5 on page 174), click **Next**.

7. In the Setup Type window (Figure 5-8), select **Typical**. The Typical installation option installs all components of Design Studio, including all of the adapters, WebSphere Transformation Extender branded ODBC drivers, and example files. Then click **Next**.

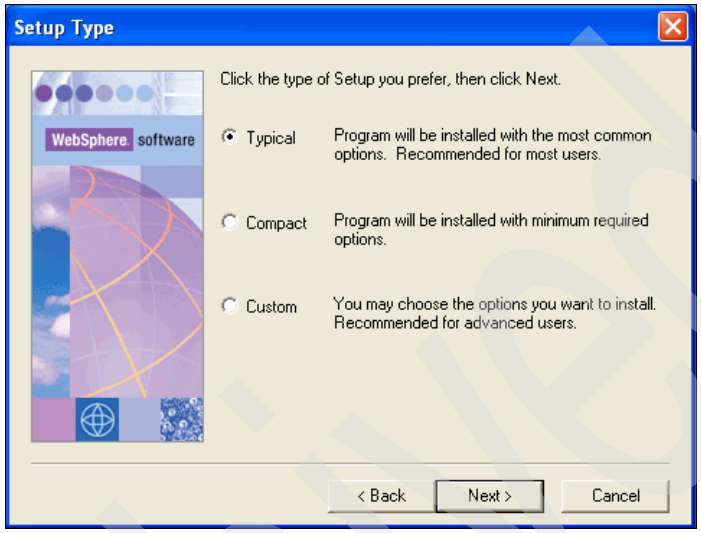

Figure 5-8 Setup Type window

8. In the Select Program Folder window (Figure 5-9), in which you can select the folder for the Design Studio programs, keep the default. Then click **Next**.

Figure 5-9 Select Program Folder window

9. In the Start Copying Files window (Figure 5-10), review the selections and click **Next** to start the installation of files.

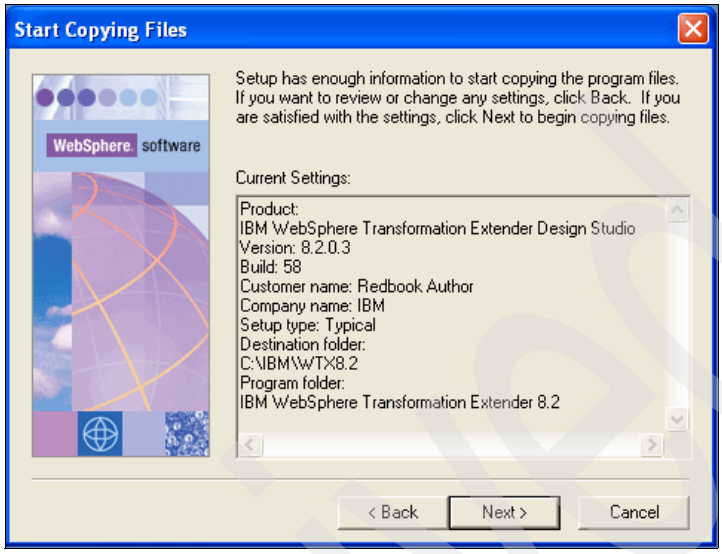

Figure 5-10 Start Copying Files window

A window opens that contains a status bar that shows the progress of the installation (Figure 5-11).

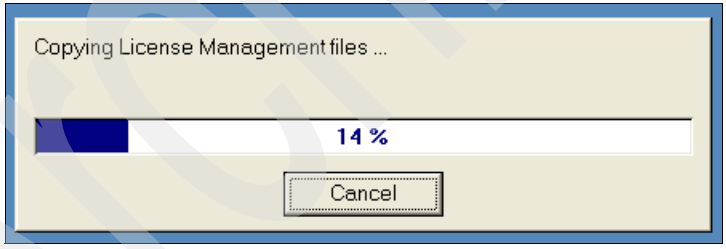

Figure 5-11 Installation progress window

10. When the installation completes, in the Setup Complete window (Figure 5-12), if you do not want to see the release notes, clear the **View release notes** check box. Then click **Finish**. The setup program closes. If you select View release notes, the file opens in your default text editor.

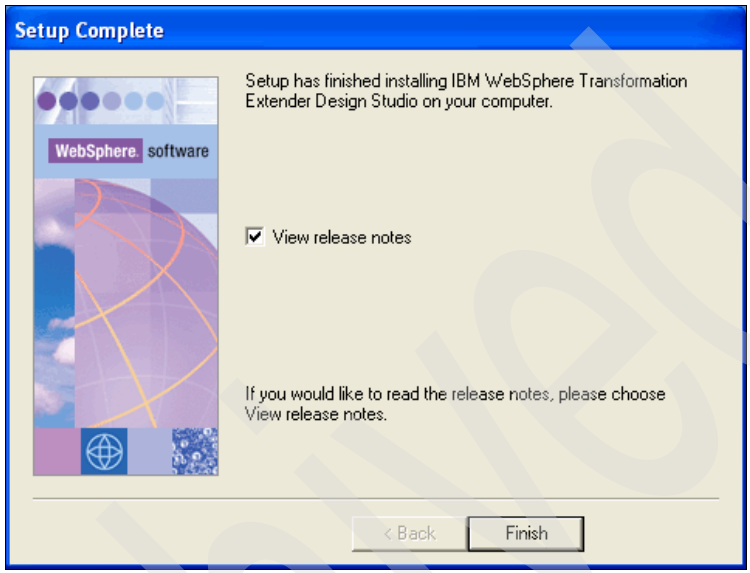

Figure 5-12 Setup Complete window

After you install WebSphere Transformation Extender Design Studio, from the Windows Start menu, when you click **All Programs**, you see the WebSphere Transformation Extender V8.2 folder (unless you changed it during installation). Click this folder to see the Design Studio folder (Figure 5-13), which contains all of the components of Design Studio.

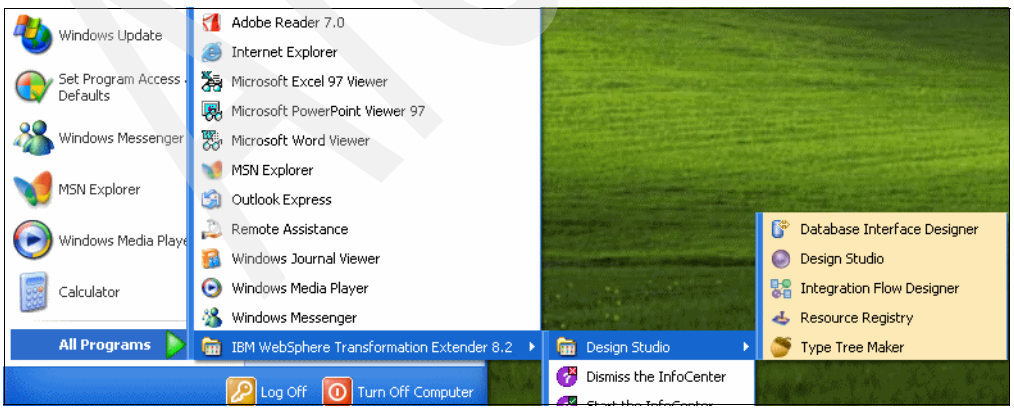

Figure 5-13 WebSphere Transformation Extender in the Windows Program folder

Adding the Transformation Extender Development perspective: If you already have the WebSphere Message Broker Toolkit V6.1.0.2 or later, or WebSphere Integration Developer V6.1.0.2 or later, installed and want to add the Transformation Extender Development perspective to your existing toolkit, you *must* perform the following actions:

1. Install the Design Studio (after the WebSphere Message Broker Toolkit or WebSphere Integration Developer).
2. Install WebSphere Transformation Extender for Integration Servers.

   After you install both of these applications, the Transformation Extender Development perspective is registered in the existing toolkit. Figure 5-14 shows the Transformation Extender Development perspective in the list of available perspectives in the WebSphere Message Broker Toolkit after the Design Studio *and* WebSphere Transformation Extender for Integration Servers are installed.

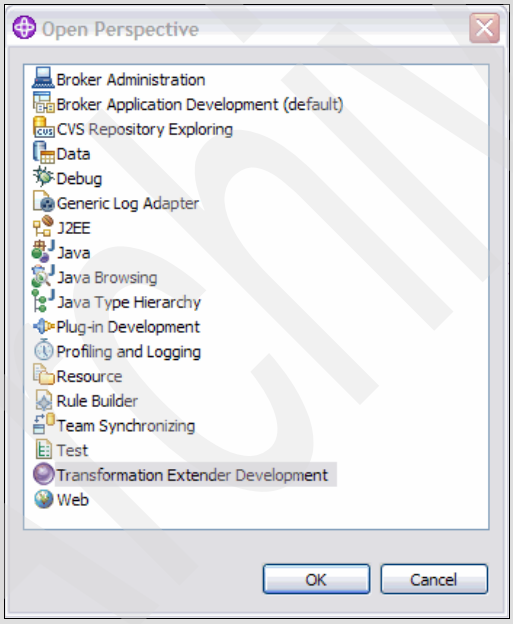

Figure 5-14 WebSphere Message Broker Toolkit available perspectives

3. After all other WebSphere Transformation Extender components and options are installed, install the WebSphere Transformation Extender Online Library.

5.2 Design Studio language

As previously explained, the language used in the Design Studio depends on the language selection of the regional settings. However, you can force a language setting as explained in the following steps:

1. In the installation directory of Design Studio, in the `dtx.ini` initialization file, uncomment line #3 by erasing the semicolon to force setup of `locale` as `en` (Example 5-1).

Example 5-1 Forcing locale as en (english) in the dtx.ini file

```
[Execution]
;specify the locale of the product (possible values
de,en,es,fr,it,ja,ko,pt_BR,zh_CN,zh_TW)
locale=en
rectLeft=92
rectTop=92
rectRight=412
rectBottom=348
```

2. In the installation directory of Design Studio, in the `DTXCommon.bat` batch file (Example 5-2):
   a. Uncomment line #15 by removing the `@rem` to force setup of `USRLANGUAGE` as `English`.
   b. Uncomment line #16 by removing the `@rem` to force setup of `USRCOUNTRY` as `US`.
   c. Uncomment line #20 by removing the `@rem` to force setup of `NLLOCALE` as `en_US`.

Example 5-2 Setup of USR language, Country, and NLLocale in DTXCommon.bat

```
@rem ==== Locale for Java Applicationa ===
set USRLANGUAGE=English
set USRCOUNTRY=US
 @rem ===================================

 @rem ==== Locale for Eclipse Applicationa ===
set NLLOCALE=en_US
 @rem ===================================
```

3. In the installation directory of Design Studio, in the `ResourceRegistry.bat` file (Example 5-3):
   a. Uncomment line #20 by removing the `@rem` to force setup of `USRLANGUAGE` as `English`.
   b. Uncomment line #21 by removing the `@rem` to force setup of `USRCOUNTRY` as `US`.

Example 5-3 Setup of USR language and Country in ResourceRegistry.bat

```
@echo off
REM ============================================================================
REM == Licensed Materials - Property of IBM
REM == 5724-Q23
REM == © Copyright IBM Corporation 1994, 2008
REM ============================================================================

set BATCH_HOME=C:\IBM\WTX8.2

if exist "%BATCH_HOME%\dtxcommon.bat" call "%BATCH_HOME%\dtxcommon.bat"

if defined DTXHOME goto :DTXHomeSet

set DTXHOME=%BATCH_HOME%

set JREDIR=%DTXHOME%\java
set JRE="%JREDIR%\bin\java" -Xmx128m
set JREBASECLASSPATH=

 set USRLANGUAGE=English
 set USRCOUNTRY=US

:DTXHomeSet

REM call %JRE% -DINSTALLDIR="%DTXHOME%" -Duser.language=%USRLANGUAGE%
-Duser.country=%USRCOUNTRY% -Dsun.java2d.noddraw=true -classpath
"%DTXHOME%\resourceregistry.jar";%JREBASECLASSPATH%
com.ibm.websphere.dtx.resourceregistry.gui.ResourceRegistry

call %JRE% -DINSTALLDIR="%DTXHOME%" -Dsun.java2d.noddraw=true -classpath
"%DTXHOME%\resourceregistry.jar";%JREBASECLASSPATH%
com.ibm.websphere.dtx.resourceregistry.gui.ResourceRegistry
```

5.3 Using Design Studio

Use Design Studio at design time to model, develop, and test your data transformation and application integration maps and systems. You can integrate both content transformation and content-based routing through the Design Studio graphical interface. Design Studio includes the following components:

Type Tree Editor Used to define data objects (including source and target data structures).

Map Editor Used to develop maps that define your data transformation logic.

Integration Flow Designer
Used to design systems of maps that model your data transformation business processes.

Database Interface Designer
Used to import metadata from relational databases to generate type trees for queries or tables.

Resource Registry Used to define resource name aliases to abstract various parameter settings that resolve to specific resources at run time.

Type Tree Maker A scripting tool that automates the capture of metadata from machine-processible sources to create graphical metadata for use in WebSphere Transformation Extender. The use of the Type Tree Maker is not discussed in this chapter. See the Type Tree Maker documentation in the Online Library for more information.

The WebSphere Transformation Extender version 8.2 Design Studio is an Eclipse-based design environment that integrates the Type Tree Editor and the Map Editor functionality. The Eclipse platform provides a common user interface for the Type Tree Editor and Map Editor called a *perspective* (a collection of views, toolbar icons, and menus that are grouped to accomplish a specific type of work).

If you are new to Eclipse, you can review information about Eclipse in the help system. To access this information:

1. Open Design Studio.
2. Select **Help** → **Help Contents**.
3. Under Contents, select **Workbench User Guide**.

To develop a WebSphere Transformation Extender solution:

1. Describe the data.

   Data that is handled by WebSphere Transformation Extender is described in type trees that define the data structure and semantics.

2. Transform the data

   Data is mapped between the source or sources and destination or destinations by using a drag-and-drop method and entering map rules.

3. Deploy the transformation.

   Compile the map and deploy it to the appropriate WebSphere Transformation Extender runtime edition.

Figure 5-15 shows a basic transformation. Centrally you can see a map (represented by the Rules box) that has one input card and one output card. Each card always links to a type tree. A type tree defines the data format, which is read (input) or written (output) through a card. The card also links the map with the adapter that is used to receive the data (the left Adapter box) or to write the data (the right Adapter box).

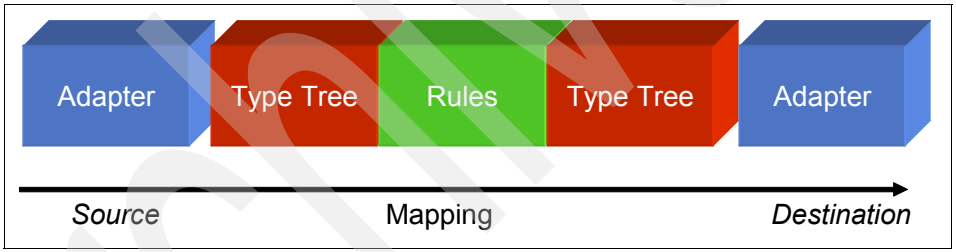

Figure 5-15 Components of a WebSphere Transformation Extender solution

The adapters that are specified at development time can be overridden after the map is deployed to the run time.

You might recall that you are not limited to a single input and single output. One of the strengths of WebSphere Transformation Extender is the ability to do many-to-many mapping with multiple inputs and multiple outputs. Figure 5-16 shows how a map can handle multiple input cards (each a type tree) and multiple output cards (again, each a type tree), each one using the adapter that is required for that data source or target.

XSD file: Remember that you can use an XML schema XSD file directly for an input card or output card instead of a type tree.

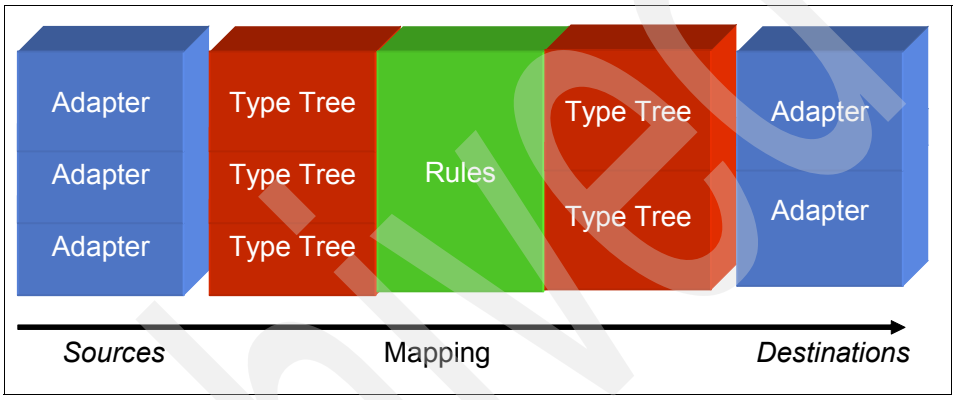

Figure 5-16 Many-to-many mapping

The first time you launch Design Studio, you see a Welcome window (Figure 5-17).

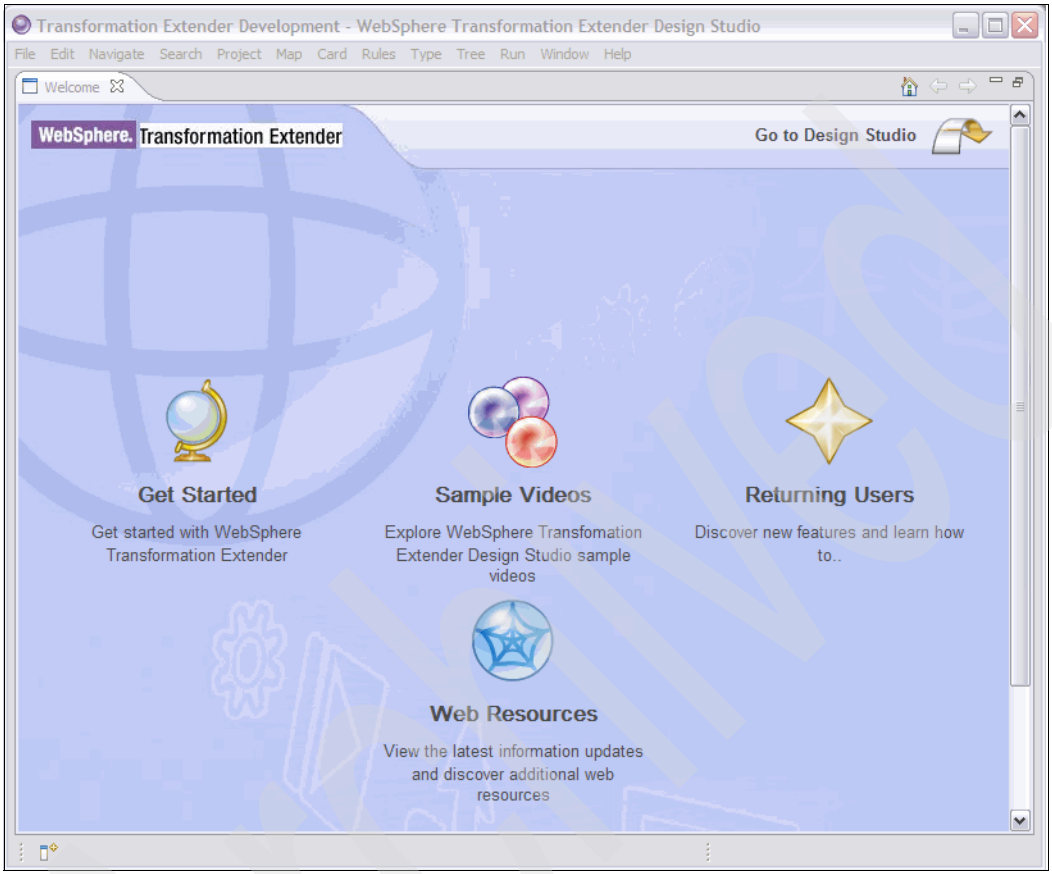

Figure 5-17 Transformation Extender Development perspective Welcome window

The Welcome window provides icon shortcuts to help you get started with WebSphere Transformation Extender. This window includes the following options:

- The “Get Started” section, which provides an introduction to WebSphere Transformation Extender
- A “Sample Videos” section for you to see videos on some basics of using WebSphere Transformation Extender, including how to load in the examples provided
- A “Returning Users” section to help you learn specifically what is new with the latest release

- A “Web Resources” section, which provides links to important Web sites, such as WebSphere Transformation Extender Support

You can close the Welcome window by clicking the **X** in the Welcome tab or by clicking **Go to Design Studio**. After you close the Welcome window, you can re-display it at any time by clicking **Help** → **Welcome** from the Design Studio menu.

Customizing the Type Tree Editor and Map Editor

You can use the Preferences settings to alter the default look and behavior of the Type Tree Editor and Map Editor within the Design Studio. To access the WebSphere Transformation Extender preferences, select **Window** → **Preferences** from the Design Studio menu. Then scroll through the list to find the Transformation Extender preferences (Figure 5-18).

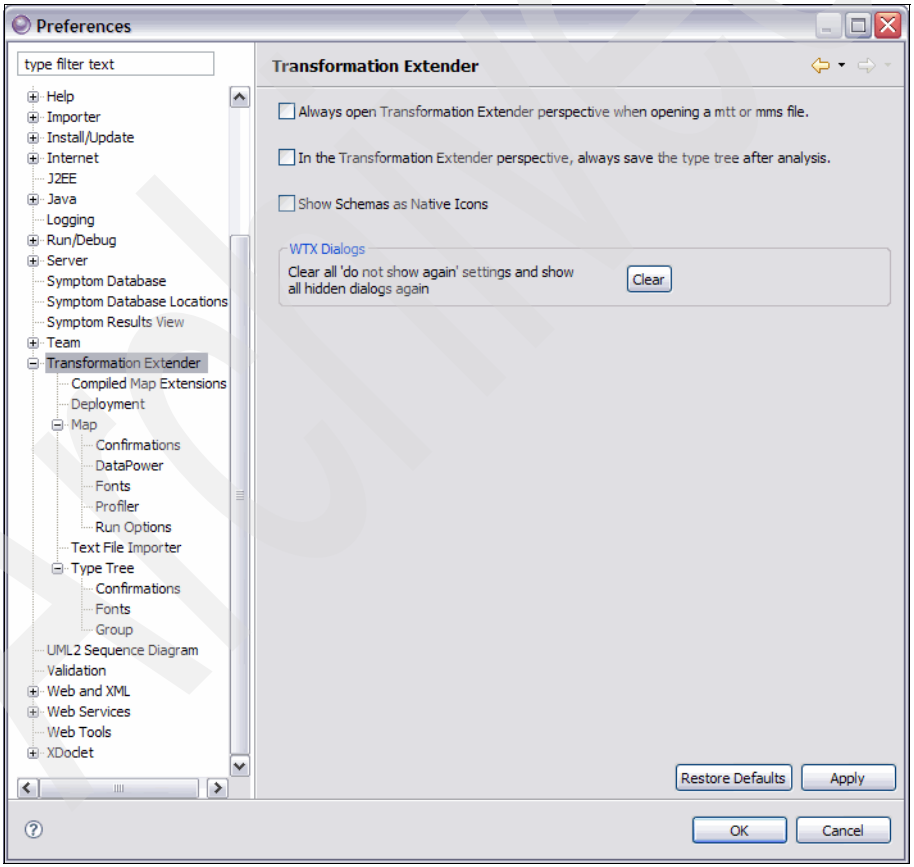

Figure 5-18 Transformation Extender preferences

On the Transformation Extender page, you see some general preferences. On the Compiled Map Extensions page (Figure 5-19), you can set the default file extensions for compiled maps on each platform.

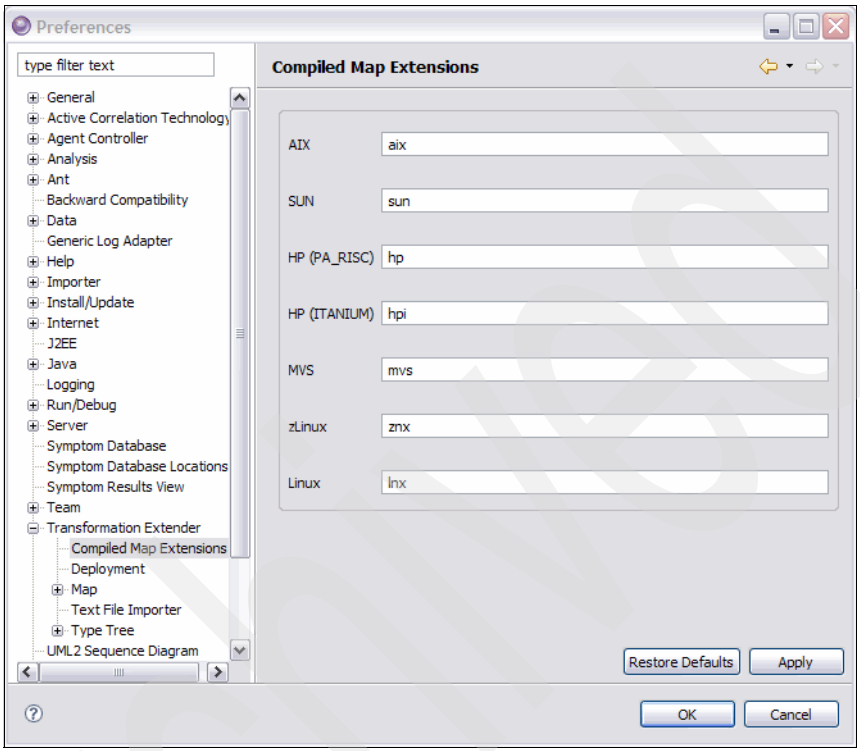

Figure 5-19 Compiled Map Extensions preferences page

On the Deployment page (Figure 5-20), you can add, edit, or remove deployment destinations.

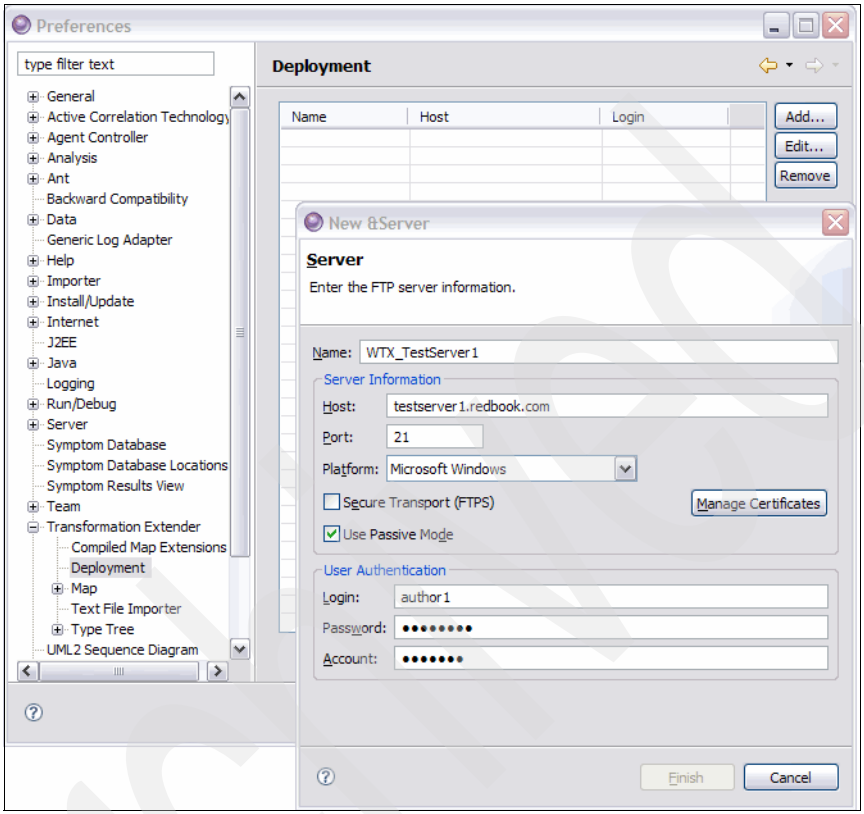

Figure 5-20 Deployment preferences page

You can customize the *Map Editor* through many options in the Preferences windows. On the top-level Map page, you customize options for saving, comparing, and viewing maps (Figure 5-21). You can set colors to color-code different parts of a map within the Map Editor. An important option here is the number of input and output cards to show within the Map Editor.

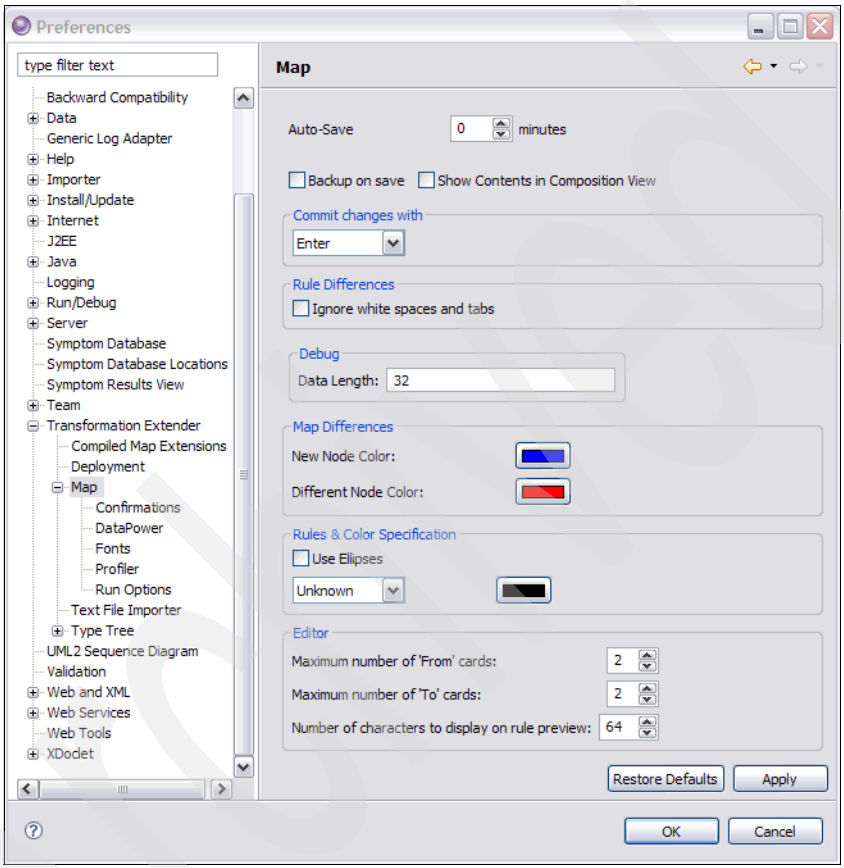

Figure 5-21 Map preferences page

On the Confirmations page (Figure 5-22), you can configure the operations for which Design Studio should prompt the user to confirm, prior to executing the operation.

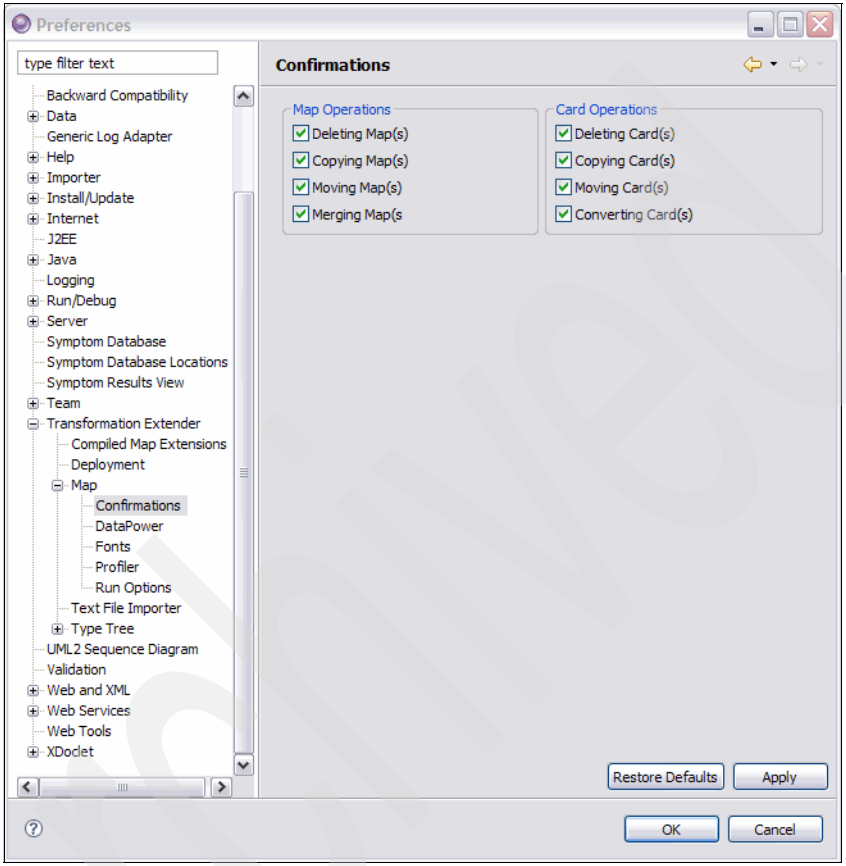

Figure 5-22 Map Confirmations preferences page

On the DataPower page (Figure 5-23), you set the location and options for interacting with a DataPower device.

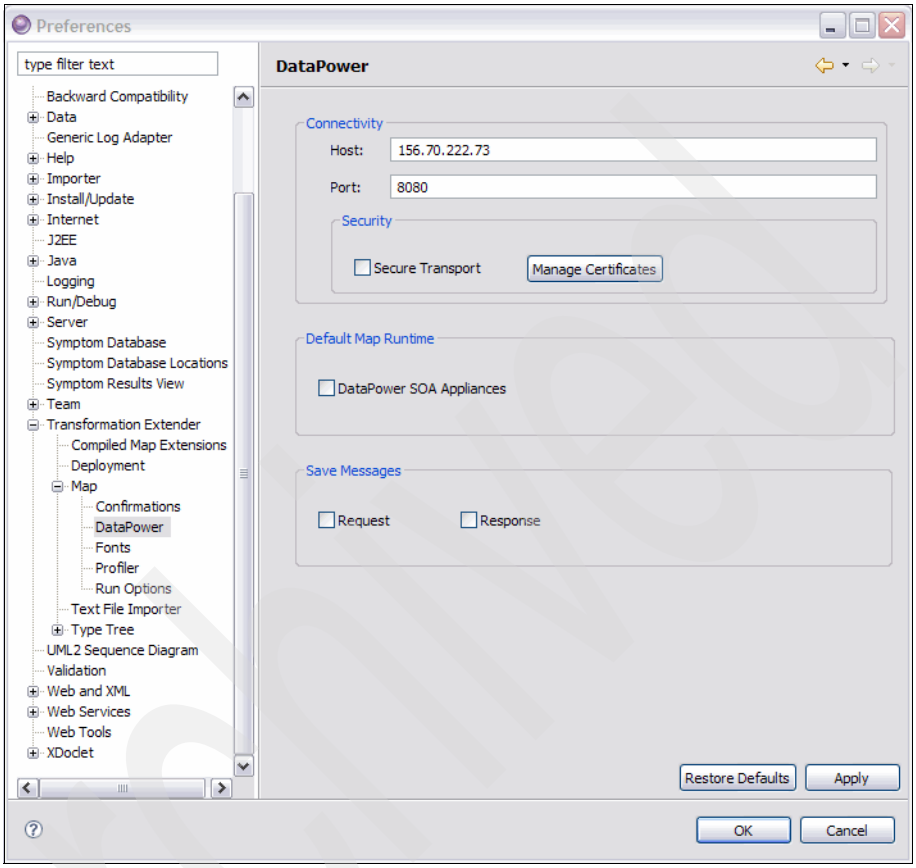

Figure 5-23 Map DataPower preferences page

On the Fonts page (Figure 5-24), you can change the font types that are used in the Map Editor views.

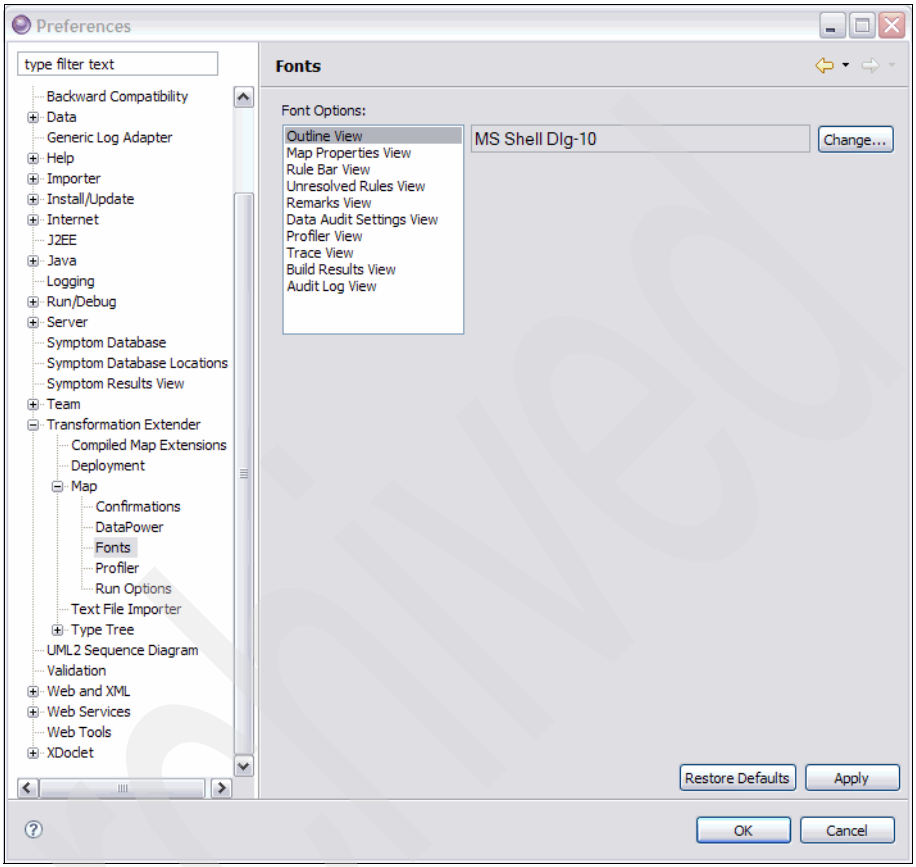

Figure 5-24 Map Fonts selection preference page

On the Profiler page (Figure 5-25), you configure the default Map Profiler utility settings. Before profiling a map, you must enable a Function Times option or a Type Times option to see statistics in the output. Otherwise the report is empty.

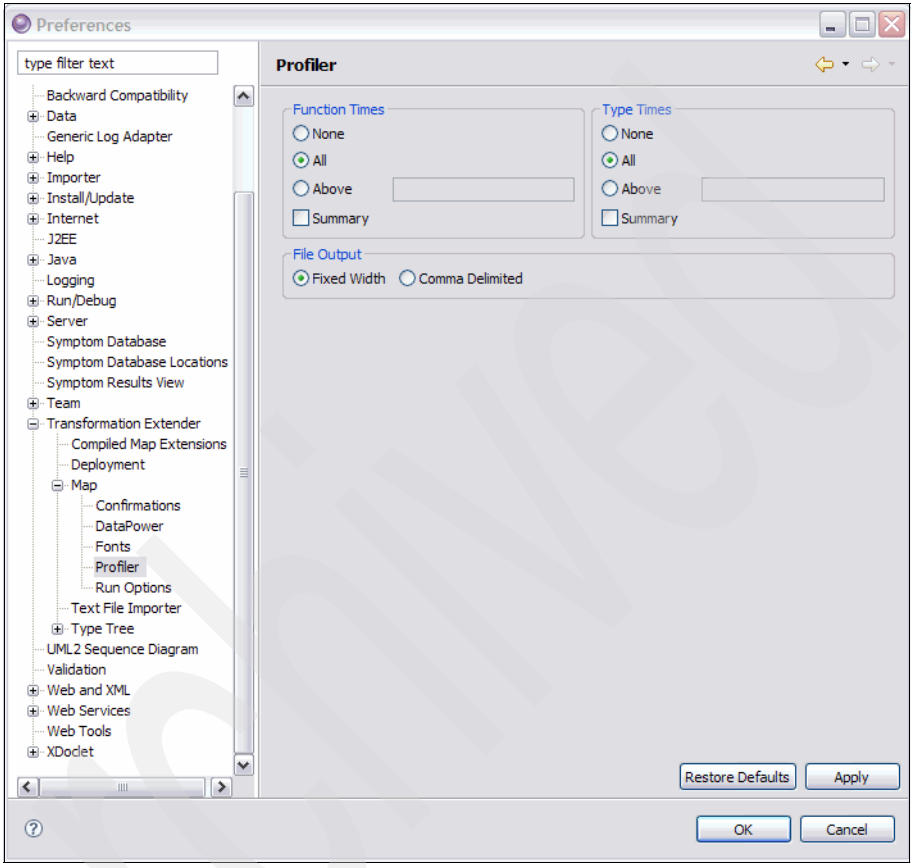

Figure 5-25 Map Profiler preferences page

On the Run Options page (Figure 5-26), you configure the default Map Run Options settings.

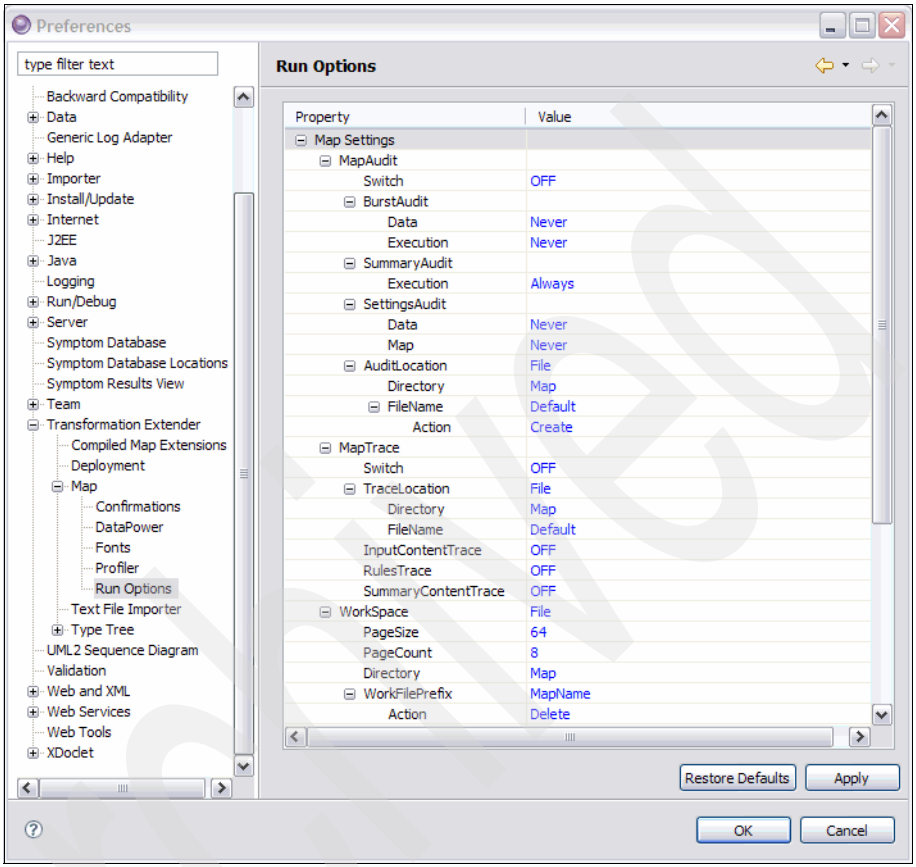

Figure 5-26 Map Run Options preferences page

On the Text File Importer page (Figure 5-27), you set options for the text file importer.

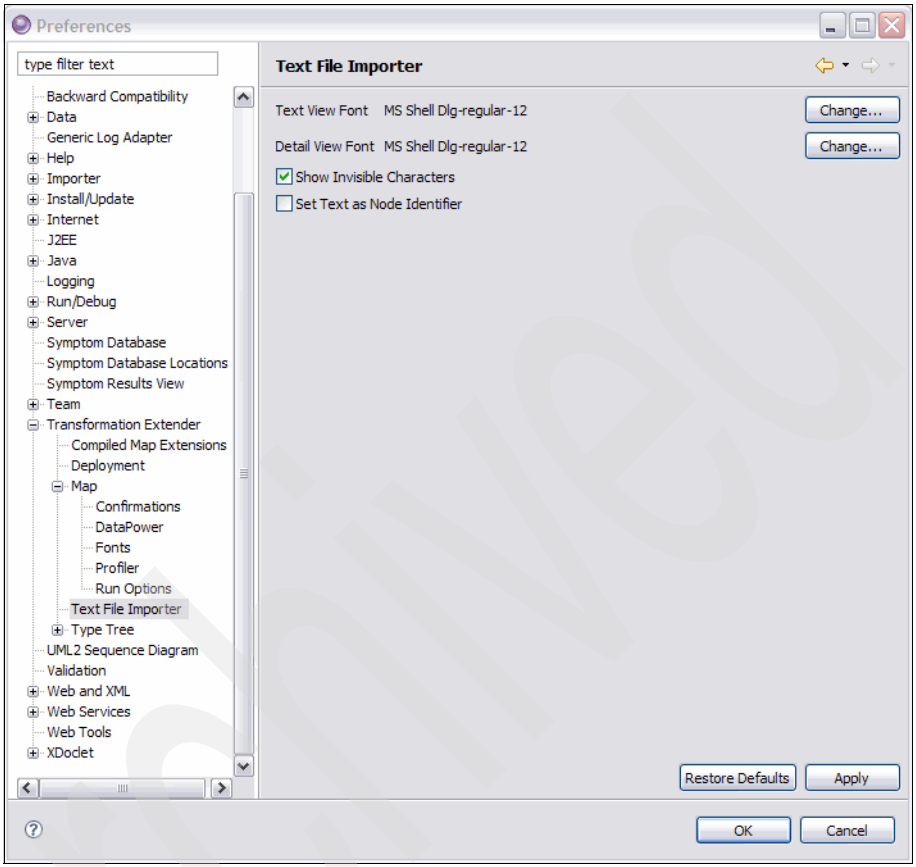

Figure 5-27 Text File Importer preferences page

You can customize the *Type Tree Editor* through many options in the Preferences windows. On the top-level Type Tree page (Figure 5-28), you customize options for saving, comparing, entering text, and setting the default root name.

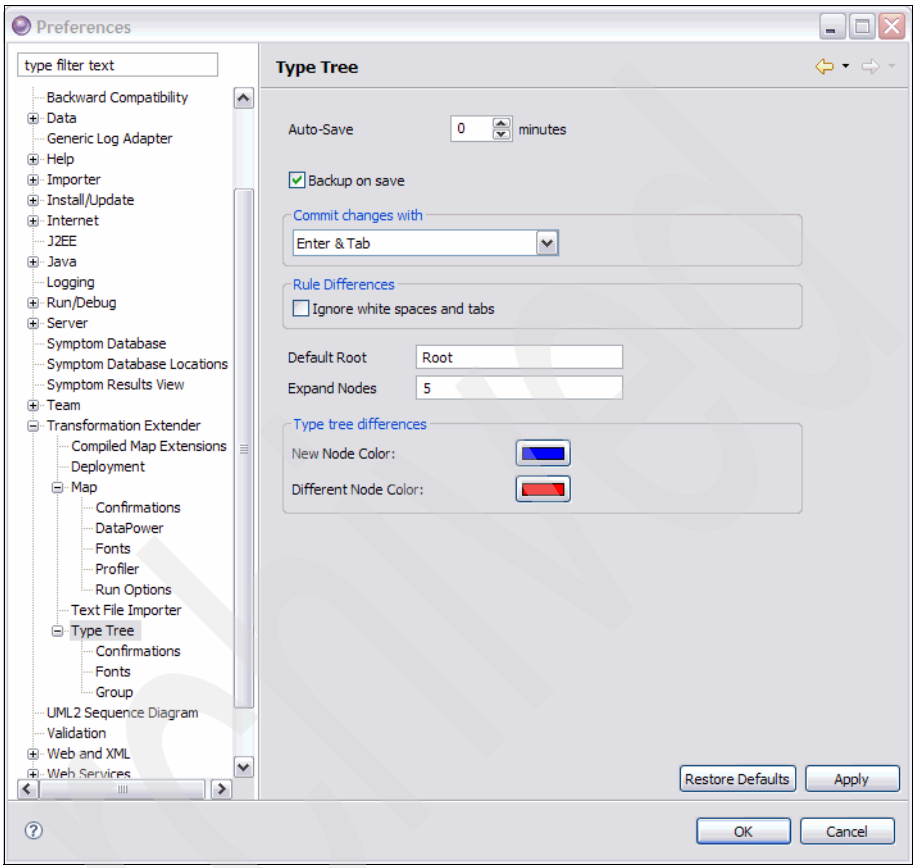

Figure 5-28 Type Tree preferences page

On the Confirmations page (Figure 5-29), you can configure for which actions the Design Studio should prompt the user for a confirmation, prior to executing it.

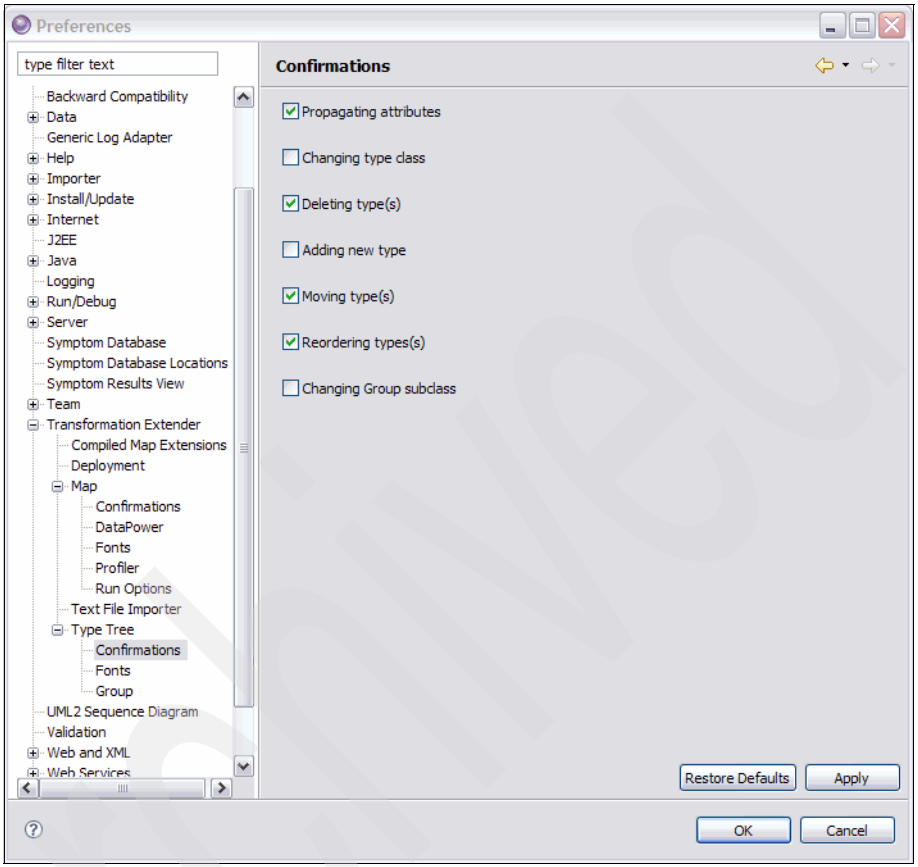

Figure 5-29 Type Tree Confirmations preferences page

On the Fonts page (Figure 5-30), you can change the font types that are used in the Type Tree Editor views.

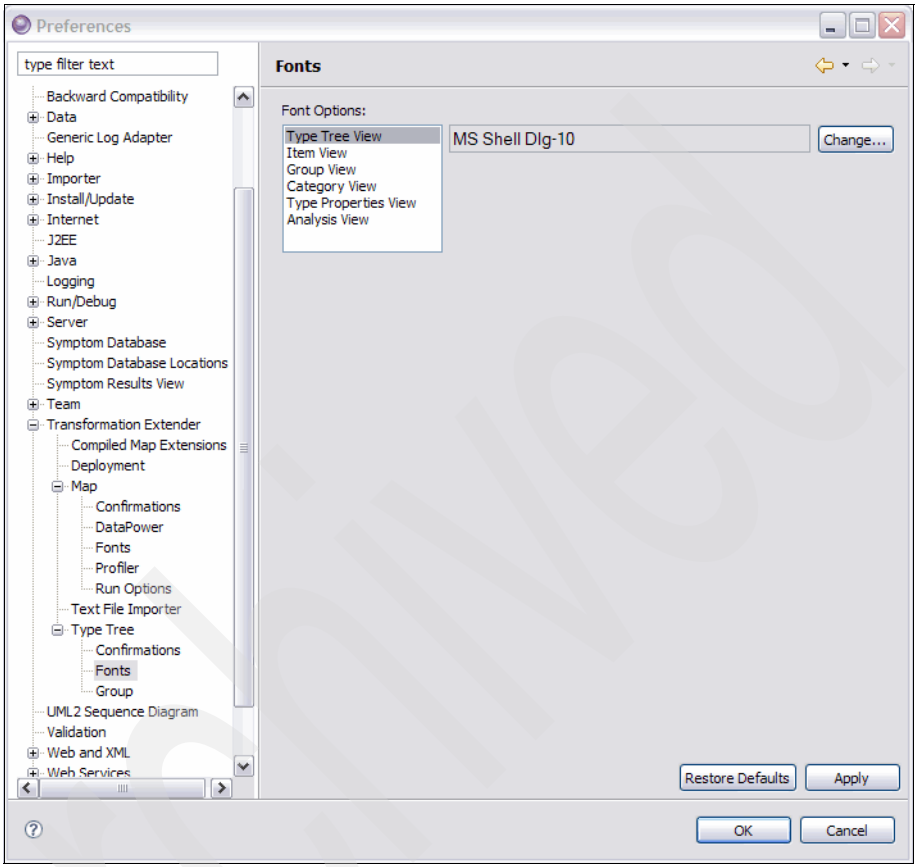

Figure 5-30 Type Tree Fonts selection preferences page

On the Group page (Figure 5-31), you customize the appearance of the group view.

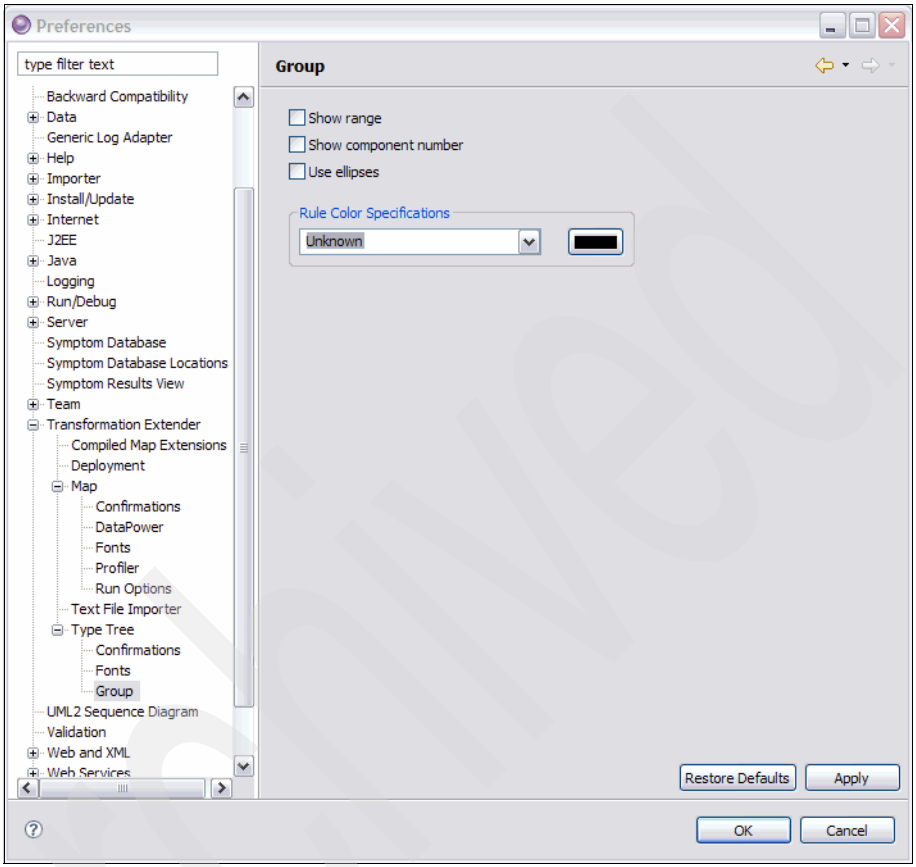

Figure 5-31 Type Tree Group preferences page

You can find a set of examples installed in the `\examples` folder under the product installation path. A variety of examples are available to help you learn more about type trees, maps, adapters, and the product in general.

The following sections take you through typical steps to create a transformation. Unless stated otherwise, the Design Studio component is used.

5.3.1 Projects

A *project* is a collection of related files (often referred to as *work files*). Projects are the largest structural unit used by the Design Studio workbench. Projects contain folders and files, and they can be opened, closed, or built. Folders can contain other folders and files. All resources are held in projects.

Projects are stored within your workspace on the file system. You can have multiple projects open at one time. Projects can be deleted from the workbench or the current workspace.

When using Design Studio, you use Transformation Extender projects. Transformation Extender projects contain structures that are common to mapping projects. An Extender Project has folders that are predefined to hold and organize the various types of WebSphere Transformation Extender design time artifacts.

To create a new Extender Project:

1. Select **New** → **Extender Project** (Figure 5-32).

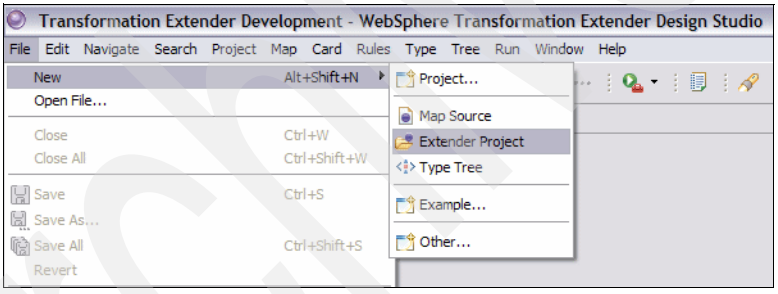

Figure 5-32 Creating a new Extender Project

2. In the WebSphere Transformation Extender Project window (Figure 5-33), enter a project name.

> **Note:** The project that you create maps to a directory structure in the file system. The default file system location is displayed in the Location field. If you want to create the project and its contained resources in a different location, clear the **Use default location** check box and specify the new location.
>
> If you want the new project to be dependent on one or more other projects, click **Next** and select the projects to be referenced.

Click **Finish**.

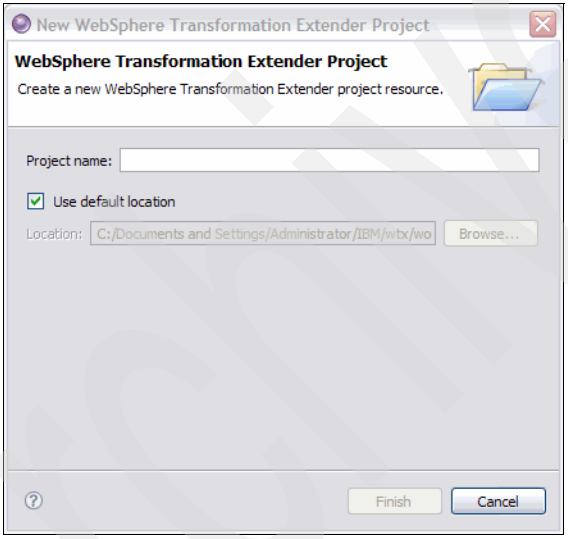

Figure 5-33 Entering a project name

The new project is listed in the Extender Navigator navigation view. The Extender project has a predefined structure as shown in Figure 5-34.

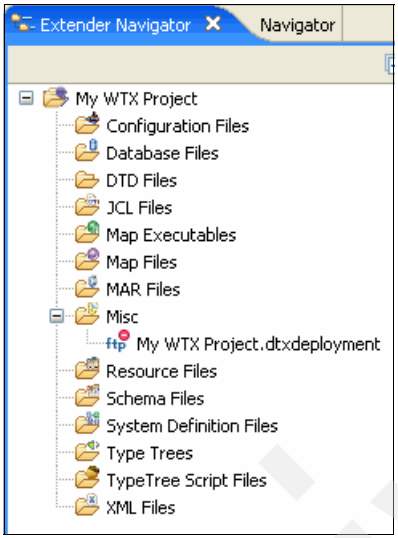

Figure 5-34 Extender Navigator view

Files and artifacts are created and stored in the project, or they are imported into the project. The project contains the following folders:

Configuration Files Resource configuration files (Resource Registry files with a .mrc extension) that are used to specify the associated resource name file and for which virtual servers are active for a map server.

Database Files Database or query files (created by the Database Interface Designer with a .mdq extension) that contain the definitions for one or more databases, as well as queries, stored procedures, and other specifications, that can contribute to the execution of a map.

DTD Files XML DTD files that are used to create a type tree (with a .dtd extension).

JCL Files z/OS job control language (JCL) files that are used to execute WebSphere Transformation Extender maps on z/OS (with a .jcl extension) and are generated when building a map for z/OS.

Map Executables Compiled map files, which are map executables that might have a .mmc (Windows), .mvs (for z/OS), .hpi (for HP-UX Itanium®), .hp (for HP-UX PA-RISC), .aix (for IBM

AIX), .lnx (for Linux Intel®), or .sun (for Sun™ Solaris™) extension.

Map Files Map source files (with a .mms extension) that maintain maps in source format and contain maps and functional maps.

MAR Files Map archive (MAR) files for WebSphere Transformation Extender for Integration Servers that are used with WebSphere Message Broker, which packages all the related artifacts into one file (with a .mar extension).

Misc All other files that do not fall into one of the other folders, such as .txt, .log, and .mtr files.

Resource Files Resource name files (Resource Registry files with a .mrn extension) that contain a named set of virtual servers and a named set of resources.

Schema Files XML schema files (with a .xsd extension) that are used for validation or to create a type tree, or directly in a map.

System Definition Files System definition files (created by the Integration Flow Designer with a .msd extension) that contain the definitions for one or more systems and the servers on which those systems are to execute.

Type Trees Type tree files (with a .mtt extension) that describe the syntax, structure, and semantics of your data.

TypeTree Script Files TypeTree script files (with a .mts extension) that are used by the Type Tree Maker. The Importer Wizard uses a series of maps to convert metadata into a type tree script file. The Type Tree Maker then processes this type tree script and generates a type tree that contains all the supported types that are defined in the imported metadata.

XML Files XML files with a .xml extension.

When you create a new file or import files, Design Studio automatically places them in the appropriate folder.

When a project is closed, it can no longer be changed in the toolkit. Also, its resources are no longer displayed in Design Studio, but they still reside on the local file system. Closed projects require less memory. Since they are not examined during builds, closing a project can improve build time.

To close a project:

1. Select the project in one of the navigation views.
2. Click **Close Project** from the menu.

To re-open the project:

1. Select the project in one of the navigation views.
2. Click **Open Project** from the menu.

One of the advantages of the Design Studio using Eclipse is the built-in support for different source control programs. When working with source control, you can create and open projects under source control, check in and out work files (artifacts), version work files, and get the latest version of a work file.

Several different source control programs are available, including PVCS Version Manager and Microsoft Visual SourceSafe®. Each program offers common and unique features that are related to protecting your source files. The Design Studio can interact with many of these programs.

5.3.2 Importing and building type trees

You can obtain type trees in several ways:

- Importing a predefined type tree from one of the Industry Packs
- Using one of the importers to generate a type tree from a COBOL copybook, XML DTD, XML schema, and so on
- Using the Database Interface Designer to generate a type tree for use with a database
- Building the type tree manually in Design Studio

Remember that you must have a project created before you can create a new type tree. When building and validating a type tree, you *must* be familiar with the specifications that define your data. To review concepts for type trees, see Chapter 3, “Basic concepts of WebSphere Transformation Extender” on page 39.

Figure 5-35 on page 205 illustrates the methodology for building a type tree. If you import a type tree or use a predefined type tree, you do not have to perform all the steps. Depending on the type of data imported, you might have to define some additional components. In some cases, you only need to analyze the type tree that was imported.

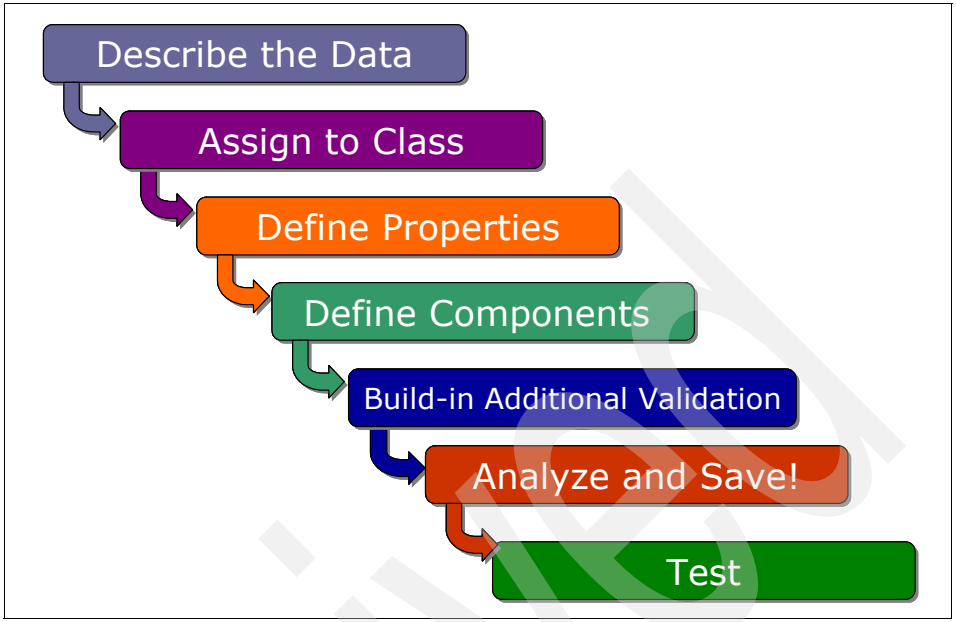

Figure 5-35 Type tree design methodology

Importing a type tree

Use the Importer Wizard to convert existing metadata into a type tree. The Importer Wizard can import various data formats to create type trees.

The Importer Wizard uses a series of maps to convert metadata into a type tree script file. The Type Tree Maker then processes this type tree script and generates a type tree that contains all the supported types that are defined in the imported metadata.

Many of the type trees that are generated by the Importer Wizard can be immediately used for map development. However, depending on the contents of the interface-specific metadata file, it might be necessary to modify the generated type tree by using Design Studio.

Important: You cannot simultaneously have a type tree open in Design Studio and import a type tree with the same name.

To create a new type tree by using one of the importers:

1. Click **File** → **Import**.
2. In the Import window (Figure 5-36), under the Transformation Extender folder, select one of the available importers to read your metadata from the list that is shown. Installing WebSphere Transformation Extender Application Packs adds an application-specific importer, such as for SAP or Siebel, to the list of available importers.

   Click **Next**.

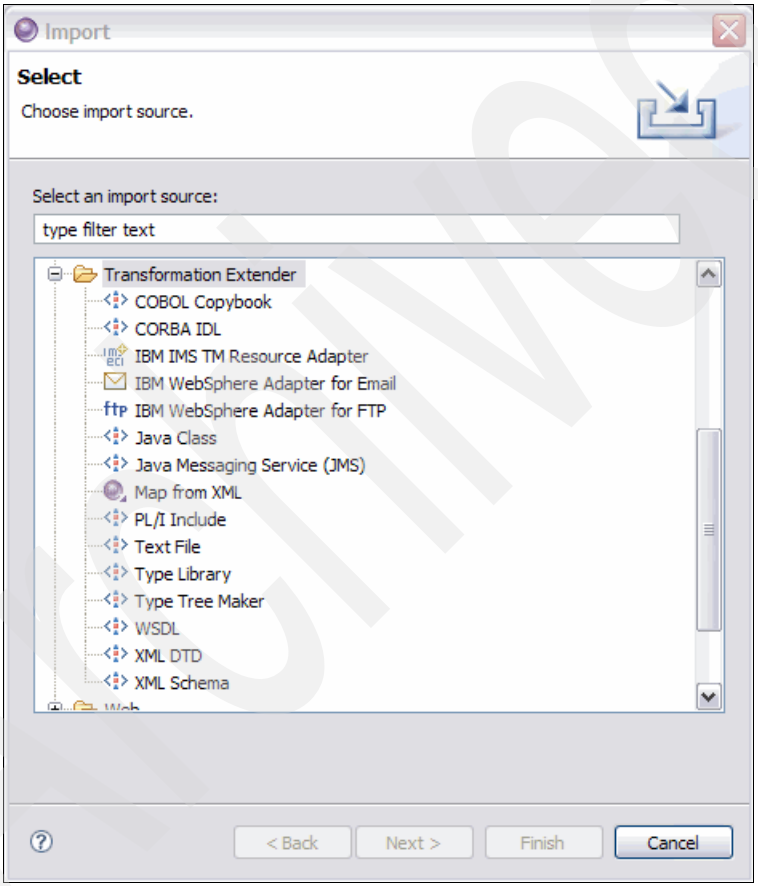

Figure 5-36 List of Importers

Several of the importers have additional windows that prompt you to specify specific options related to the metadata. After the importer completes, the new type tree is stored in the project and you are given the option to open it for your review.

In the following example, we explain how to create a type tree by importing a COBOL copybook:

1. Select the Type Tree folder within the project where you want to create the new type tree, right-click, and select **Import** → **COBOL Copybook** (Figure 5-37).

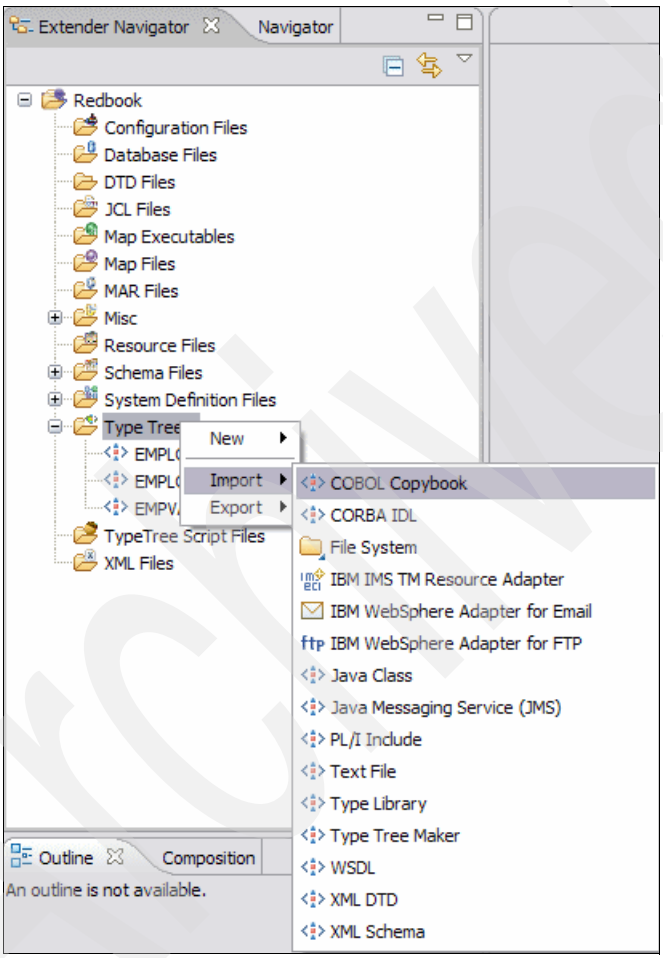

Figure 5-37 Importing the COBOL copybook

Tip: As a shortcut to start the import process, you can right-click the project name and select **Import**.

2. In the COBOL CopyBook Importer window (Figure 5-38), where you are prompted for the source copybook file, enter the path to the file or click the ellipses button (...) to graphically choose the file. The importer looks for copybooks with a .cpy extension (or you can select any file). Select the file and click **Next**.

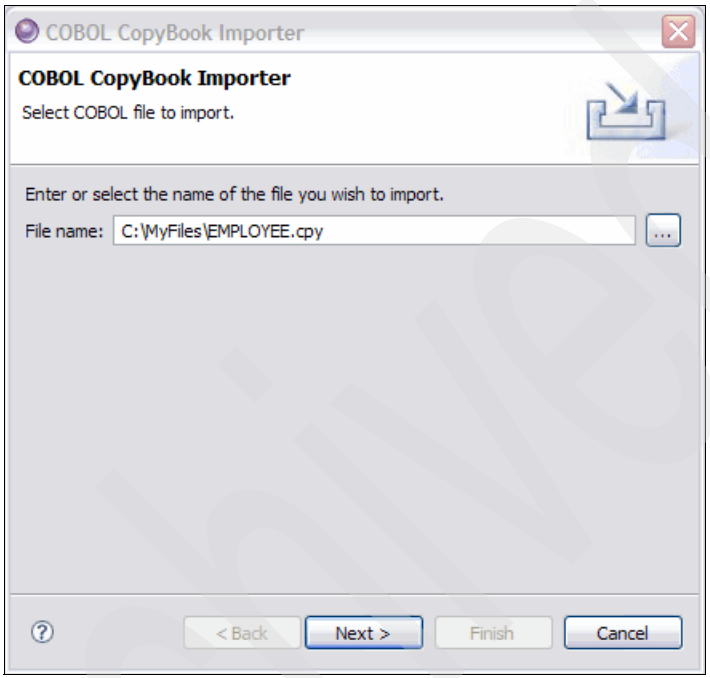

Figure 5-38 Selecting the file to import

3. In the next window (Figure 5-39), select the byte order and character set information. You can choose between EBCDIC, ASCII, and NATIVE. By selecting NATIVE, the platform can dictate the correct implementation, making the type tree portable. Click **Next**.

Figure 5-39 Selecting the byte order and character set

4. In the next window (Figure 5-40), select the file name of the type tree and the location to store the type tree. A name is suggested based upon the Copybook file name. Then click **Next**.

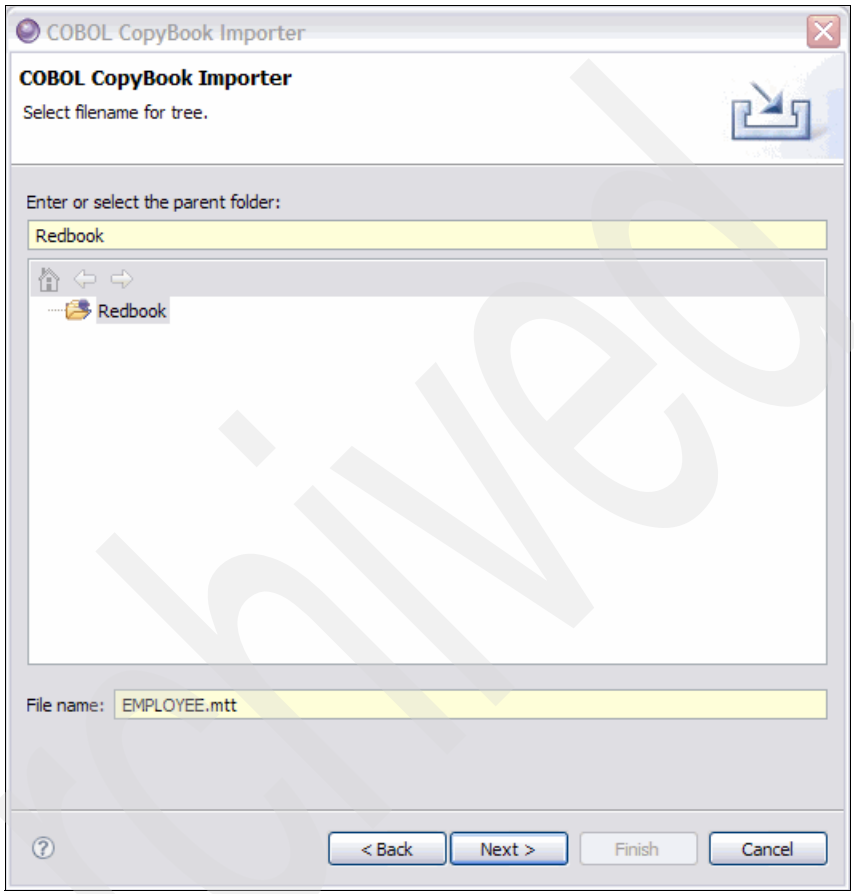

Figure 5-40 Selecting the type tree name

The Copybook Importer reads the copybook and attempts to create a type tree based on its contents. At the bottom of the window, you see a Creating type tree progress indicator, which shows the importer's progress until the process is complete.

5. Upon completion, in the next window (Figure 5-41), review the summary of the import process, which lists the number of warnings or errors that are encountered and a description of each. Click **Finish**.

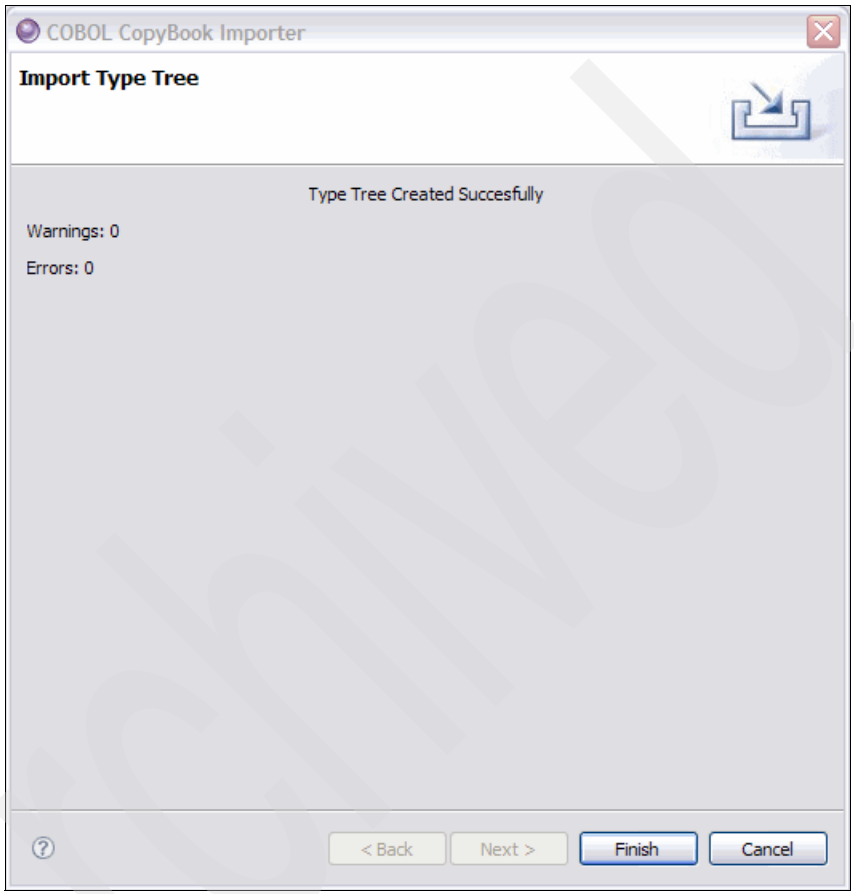

Figure 5-41 Type tree created successfully

6. If your type tree was created successfully, in the dialog box that prompts you to open the type tree now (inset in Figure 5-42), click **Yes** or **No**. You can open it to review it and make any additional modifications. For example, when importing a copybook, you might want to create a FILE level group to represent the entire file of records that the copybook represents.

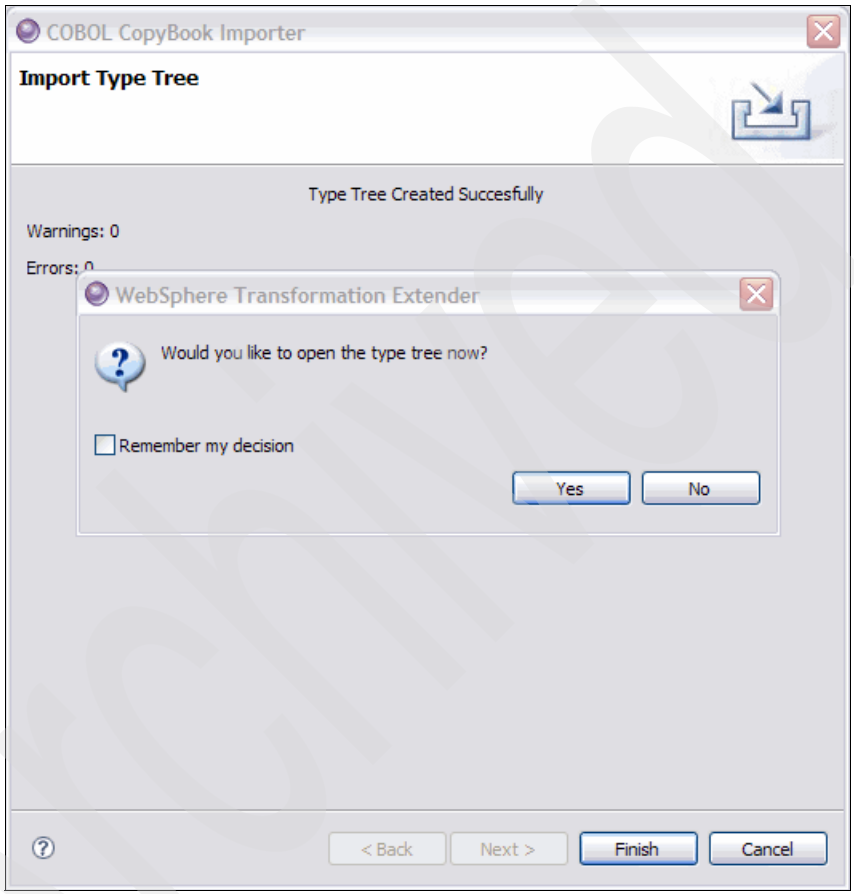

Figure 5-42 Selecting to open the type tree

Figure 5-43 shows a sample type tree created from a simple copybook.

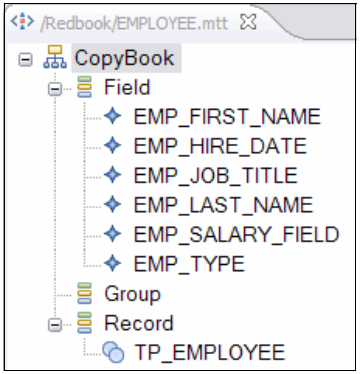

Figure 5-43 Type tree created from COBOL Copybook

In Chapter 9, “Integration scenario: WebSphere Transformation Extender on System z” on page 623, we show some examples on how to create type trees by using different importers.

Building a new type tree

To create a new type tree:

1. From the Design Studio menu (Figure 5-44), select **File** → **New** → **Type Tree**, or right-click the Type Tree folder in your project and select **New** → **Type Tree**.

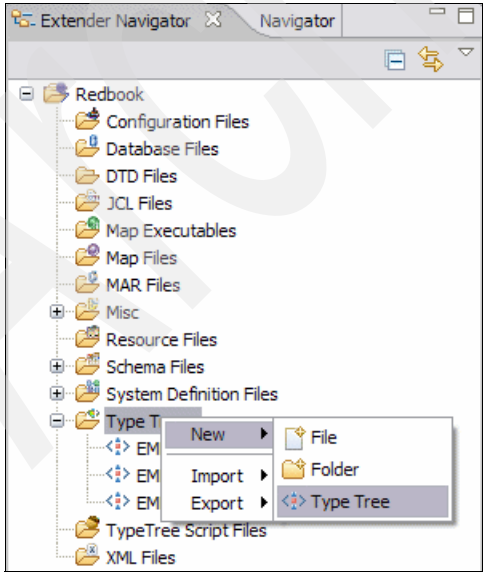

Figure 5-44 Creating a new type tree

2. In the next window (Figure 5-45), select the project for the type tree and enter a type tree name. Click **Finish**.

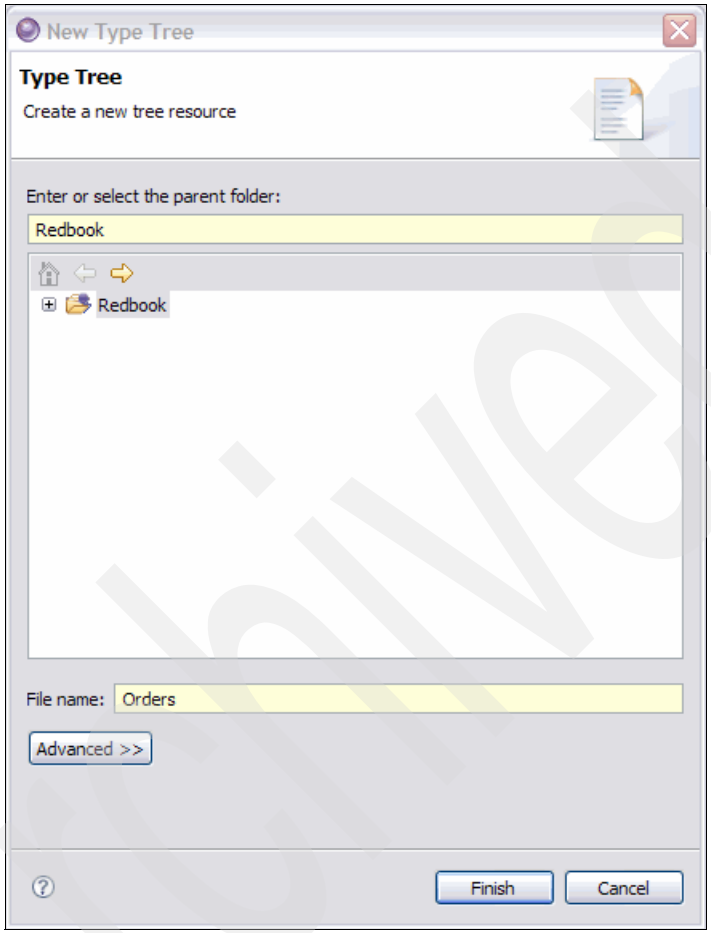

Figure 5-45 New type tree

The new type tree is displayed under the project folder in the Extender Navigator. In the main view, a tab has been created for the new type tree from where you can begin adding types that define your data to the Root type.

The following list outlines the process for creating type trees:

1. Identify the data objects in your data, and define each piece of data that you intend to map.
2. Create types for each data object in your data.
3. Define the properties of each type.
4. Create component lists.
5. Define component rules, if needed.
6. Define item restrictions, if needed.
7. Analyze and save the tree.

Types are always created as a subtype of the currently selected type. A new type tree has a single root type. All types are added as subtypes of the root type and then as subtypes of other types.

To add types to the type tree:

1. Select the type under which you want to add a type. For example, select **Root**.
2. Use one of the following methods to add a type:
   - Click **Insert**.
   - Right-click and choose **Add**.
   - From the Design Studio menu, select **Type** → **Add**.

   The program prompts for confirmation unless that setting has been disabled in the user preferences.
3. If prompted, click **Yes** to confirm that you want to add a type.
4. Enter a name for the type.
5. Open the properties for the type. Select **Type** → **Properties**.
6. Define the type properties. The most used properties are discussed in the following section.
7. Save the changes.

For optimal data transformation, the entire contents of your data must be defined. By using Design Studio, you can define input data so that each data object of the source data is identified. You can define the output data according to your output specifications.

Adding a new type to a type tree: When adding a new type to a type tree, keep in mind the following points:

- The type name is case sensitive. It can contain numbers and special characters, but not spaces.
- The words ANY, COMPONENT, FALSE, IN, LAST, NONE, TRUE, WHERE and <space> are reserved by Design Studio and cannot be used.
- Although LAST is a Design Studio reserved word, it is possible to create data types named “LAST,” “Last,” and “last.”
- A type name can have up to 256 characters. However, it is better to use short type names since the full path name is used in the mapping rules. Instead of a long name, use the Description field to explain the type.
- A type name cannot begin with a special character other than the number sign (#), tilde (~), percent (%), underscore (_), single quotation mark ('), accent (`), semicolon (;), period (.), question mark (?), and backslash (\).

The default sequence for subtypes in a type tree is in ascending alphabetic order. The Order subtypes property for a type can change that to be ascending, descending, or a custom sequence, as shown in Figure 5-46.

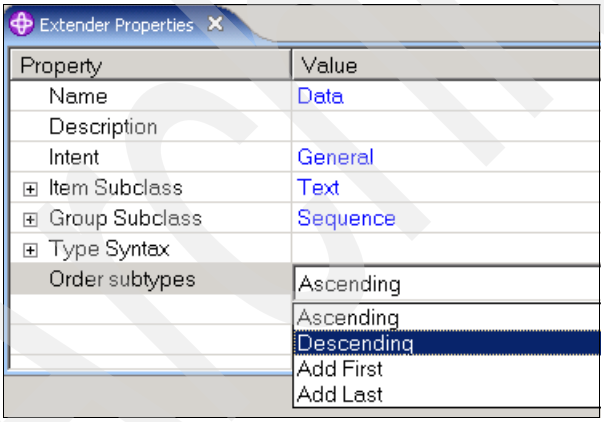

Figure 5-46 Order subtypes property

Type properties

Name and Class are the two basic properties that need to be defined for a type. In addition, you might want to include a description that explains the purpose of the type. Most of the other properties are context sensitive, that is, depending on the class and subclass of the type. For general concepts about the type properties, see Chapter 3, “Basic concepts of WebSphere Transformation

Extender” on page 39. In the next paragraphs, we present the most common properties.

Date and Time formats

For an item of the Date and Time subclass, you must define the Date format, Time format, or both. You can choose among the predefined formats or create a custom format.

To create your own date format, time format, or both:

1. In the Properties view for the Format property, click the ... button in the Value column.
2. In the Data Time format window (Figure 5-47), from the Date list, select **Custom**. Then click the ... button.

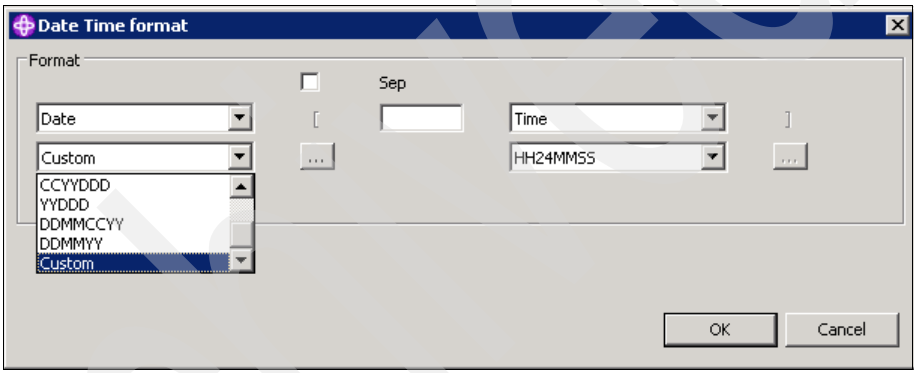

Figure 5-47 Date Time format window

3. In the Data Format window (Figure 5-48), define the custom date format. Click **OK**.

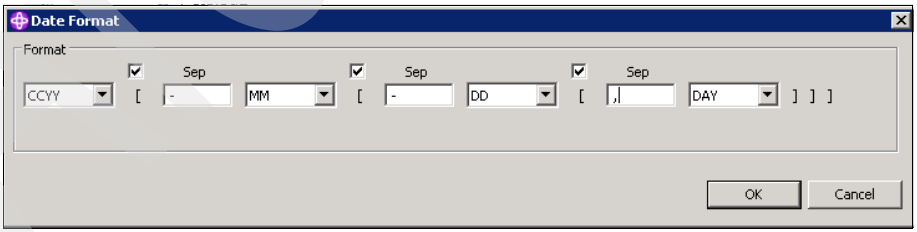

Figure 5-48 Date format customization window

4. In the Date Time format window, click the **OK** button. The custom format is displayed in the Properties view.

Number formats

If the number has a separator, set Separators to **Yes**. Select the format from the list and define the separator syntax, as shown in Figure 5-49. Separators enclosed in square brackets ([ ]) indicate that the separator is optional.

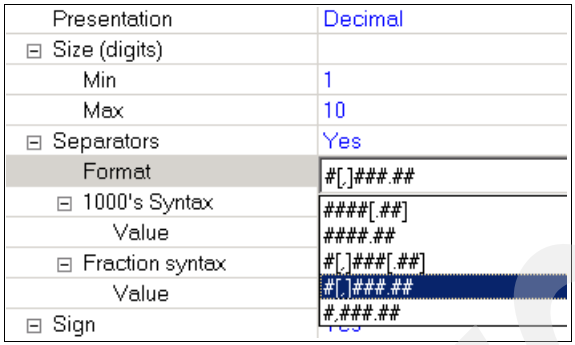

Figure 5-49 Decimal format choices

You can also define the Sign properties. Select a predefined value or specify a custom value for leading and trailing signs, as shown in Figure 5-50.

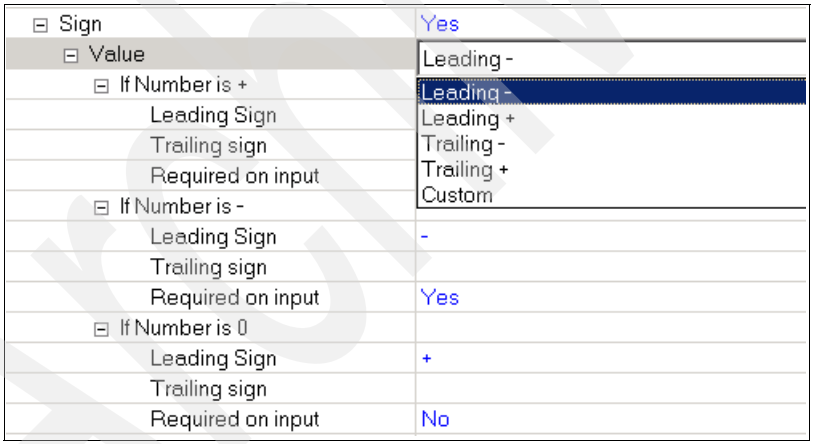

Figure 5-50 Sign property choices

When a decimal number has no separator, use the Places property to specify the number of implied decimal places.

Type Syntax property

Under the Type Syntax property, you define the Initiator, Terminator, Release, and Empty as none, literal, or variable.

To define an Initiator, Terminator, or Release as *literal*:

1. Select **Literal** from the property list.
2. Enter the exact literal value without surrounding quotation marks or additional characters. You can also use the Symbols window for non-printable characters, as described in “Inserting nonprintable characters.”
3. Specify the national and data language.

To define an Initiator, Terminator, or Release as *variable*:

1. Select **Variable** from the property list.
2. Enter the exact literal value without surrounding quotation marks or additional characters. You can also use the Symbols window for non-printable characters, as described in “Inserting nonprintable characters.”
3. From the Item list, select the syntax item.
4. Indicate whether the variable should be determined for each occurrence of the object by selecting **Yes** from the Find list.

Inserting nonprintable characters

You can insert nonprintable characters when you define the value of Pad, Initiator, Terminator, and Release Character properties:

1. To open the Insert Symbols window, click the **Value** column in the Properties View. Click the ... icon
2. In the Insert Symbols window (Figure 5-51 on page 220), select the symbol that you want to use and click **Insert**. You can select several symbols by repeating this procedure.

Important: When inserting special symbols, keep in mind the following points:

- To provide platform independence, the New Line symbol <NL> can be used in place of <CR><LF>, <LF> or HEX15.
- The whitespace <WSP> symbol can be any combination of <SP>, <HT> and <NL> and is built as one space.
- If required, a hex value enclosed in double angle brackets can be typed directly into the value field. For example, <<00>> indicates a hex null value, and <<09>> indicates a hex hard TAB.

Then click **OK**.

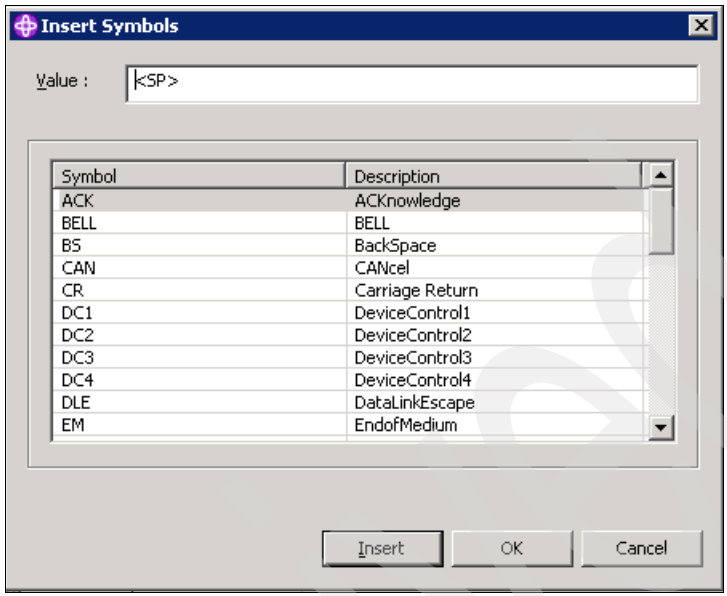

Figure 5-51 Insert Symbols window

Propagating type properties

After the type is created, the properties of that type are independent of the properties of its supertype. Changing the properties of a type does not automatically change the properties of its subtypes.

If you want to resynchronize the properties of a type with its subtypes, use a property's Propagate option. Propagation passes down the settings of a particular property from a given type to all types in its subtree.

To propagate properties:

1. Open the Properties view for the type with the properties to propagate.
2. Select the property to propagate in the Properties window. To select multiple properties, press and hold down the Ctrl key while selecting the properties. Right-click and select **Propagate** (Figure 5-52).

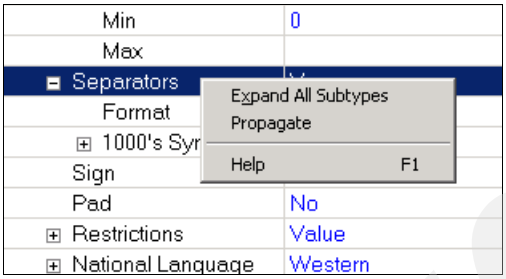

Figure 5-52 Selecting the Propagate action

3. In the confirmation window that opens, click **Yes**.

Moving a type in a type tree

If you add a type to a type tree, and later decide you want it in another place, you can move the type to another location. To move a type in a type tree:

1. Select the type to be moved, right-click and select **Move**.
2. Click the desired location and then click **Move**.

Components

After you add all the types to a type tree and define their properties, you define the components of the groups elements in the type tree. By defining the components, you are describing the order in which each type is displayed in the data stream.

To define a component:

1. Open a component by using either of the following methods:
   – Select the group or category, right-click, and select **Open** (Figure 5-53).
   – Double-click the group or category.

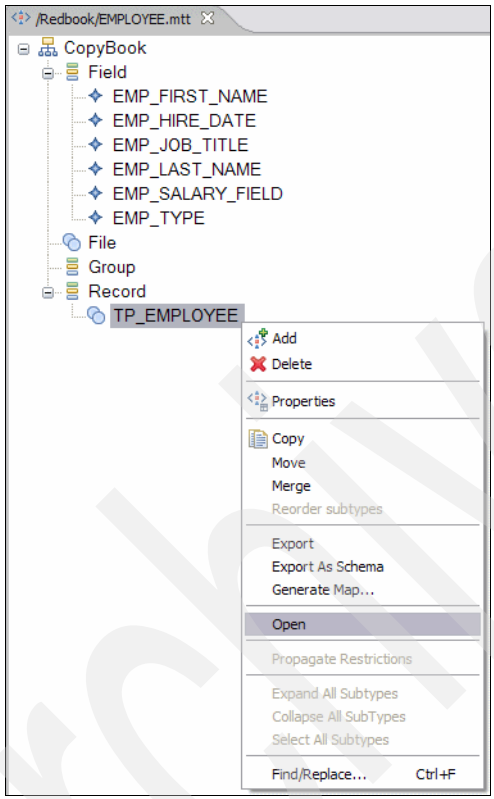

Figure 5-53 Opening a component

A blank component view opens for that group or category (Figure 5-54).

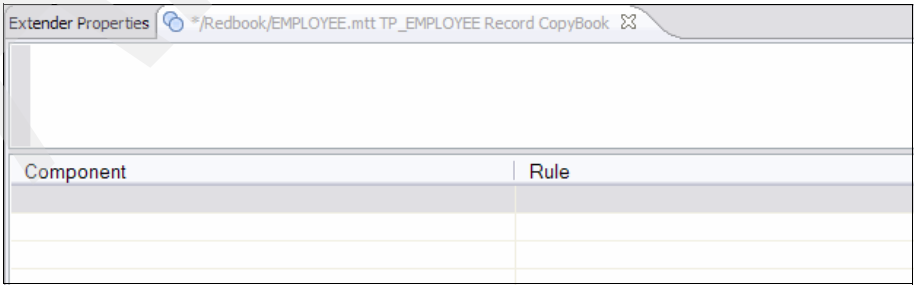

Figure 5-54 Component view

2. Drag the objects (items, other groups, or both) to the component view to indicate the order in which the components are displayed in the data (Figure 5-55).

Important: Always drag the components to the component lists, instead of typing their names for the following reasons:

- This method avoids typographical errors. Type names are case sensitive. Therefore, you *must* enter them in the exact case in which they were created.
- This method avoids incorrectly entering a relative type name. A type is identified by its full name. For example, LineItem is a subtype of Number, which is a subtype of Field. Its full type name is LineItem Number Field.

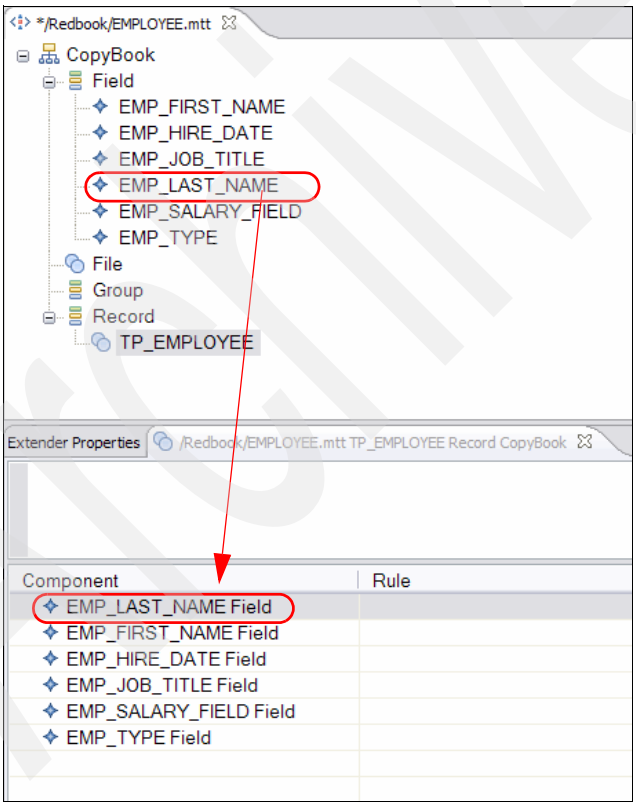

Figure 5-55 Populating the component view

3. Save the component by either pressing CTRL+S or selecting **File** → **Save**.

Each component in the component list must be interpreted as a unique object. Therefore, each component must have a unique name. If you need one object to appear more than once, you can set the range for each item in the component. The component range specifies the number of *consecutive* occurrences of that component that may appear at that point in the data stream.

To set the range of a component:

1. Right-click the component and select **Set Range** (Figure 5-56).

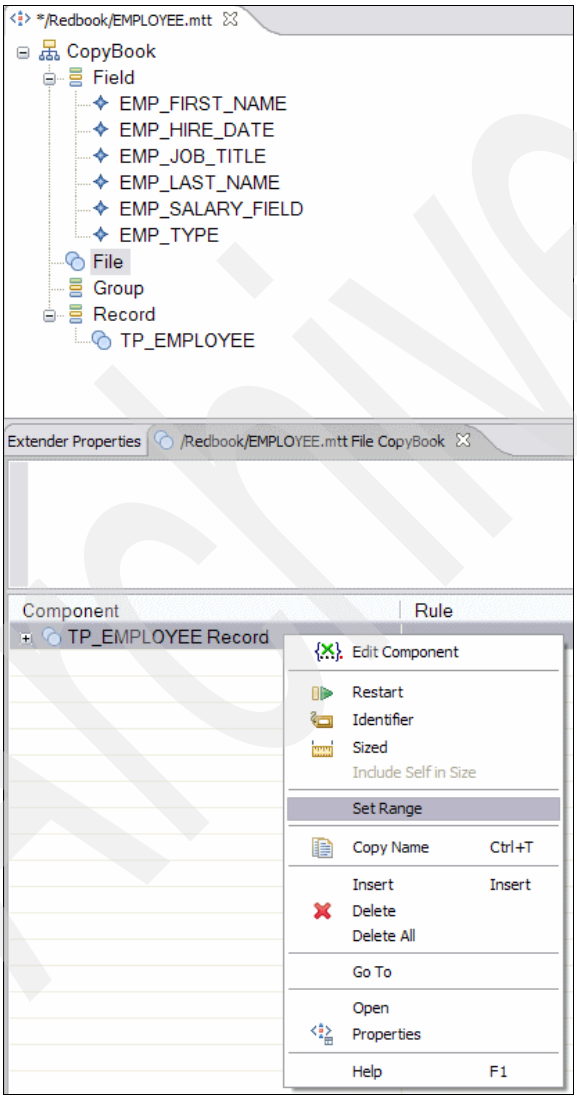

Figure 5-56 Setting the component range

2. In the Set Range window (Figure 5-57), set the appropriate Min and Max values. The default is Min (minimum of) 1 and Max (maximum of) 1 instance. Click **OK**.

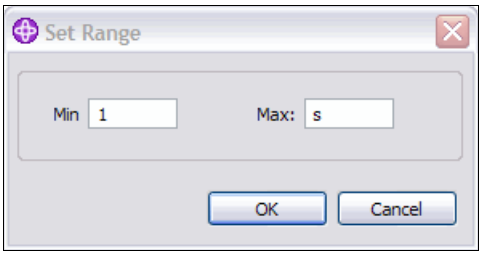

Figure 5-57 Set Range window

As shown in Figure 5-58 on page 226, the component range is then displayed after the component name in parentheses and is specified as two numbers separated by a colon (Min:Max):

- The first number is the *minimum number* of consecutive occurrences of that component. This value is optional. If it is not specified, it is interpreted as a minimum of 0.
- The second number is the *maximum number* of consecutive occurrences of that component.

To specify an indefinite number of occurrences of a component, specify the maximum as “s”, which stands for “Some”.

Showing the range after the component name: You can choose whether you want the range to be shown after the name of the component by changing the preferences for the type tree. In the menu bar, select **Window** → **Preferences** → **Transformation Extender** → **Type Tree** → **Group**.

Changing the range of multiple components: You can change the range of multiple components at the same time:

- To select a contiguous range of components, select the first component, press and hold the SHIFT key and then select the last component.
- To select non-contiguous components, select the first component, press and hold the Ctrl key and then select the other components.

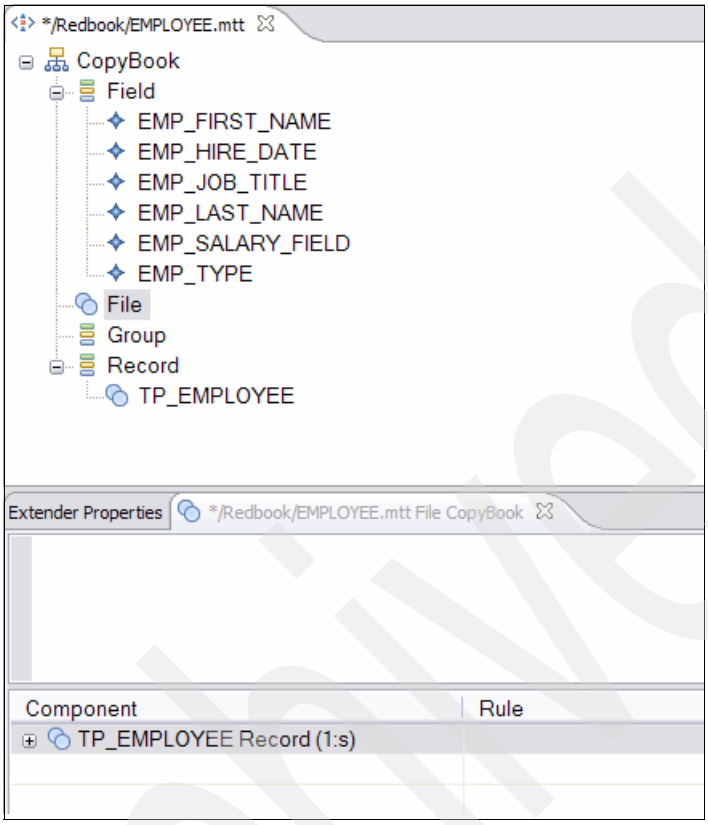

Figure 5-58 Component range view

Validation capabilities

Sometimes it is necessary to validate more than the format of a field, for example, when data for a given item is only valid if it is one of a specific set of values. This kind of validation can be done in the type tree, so that you do not need to check the values when doing the transformation. A type tree can be designed to compare the data to specified values and only allow data that meets the validation criteria to pass to the next stage in the mapping process. This eliminates time spent processing invalid data.

The additional validation can use the following capabilities:

- *Restriction lists* to limit an item to a particular value or set of values
- *Component rules* to specify conditions that must be met for a particular component to be valid

Restriction lists

With restriction lists, you can validate data when an item is limited to a specific set of values. However, you must consider the following issues when deciding to use a restriction list:

- Restriction lists are case-sensitive by default. If the data object in the data stream can come in any case (uppercase, lowercase, or mixed case), you must set it in the Properties view for the type to ignore the case. To do that, under Restrictions, set Ignore case to **Yes** (Figure 5-59).

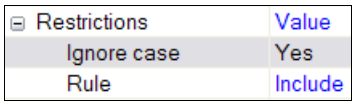

Figure 5-59 Ignore case option for restriction lists

- Every time the restriction list is changed, you must analyze and save the type tree, as well as compile and test the map.

If you want the map to ignore all restriction lists, change the map settings properties:

1. Open the map for which you want to change the properties.
2. In the menu bar, select **Map** → **Map Settings**.
3. From the Validation menu, select **Custom**.
4. Change RestrictionError to **Ignore**, as shown in Figure 5-60.

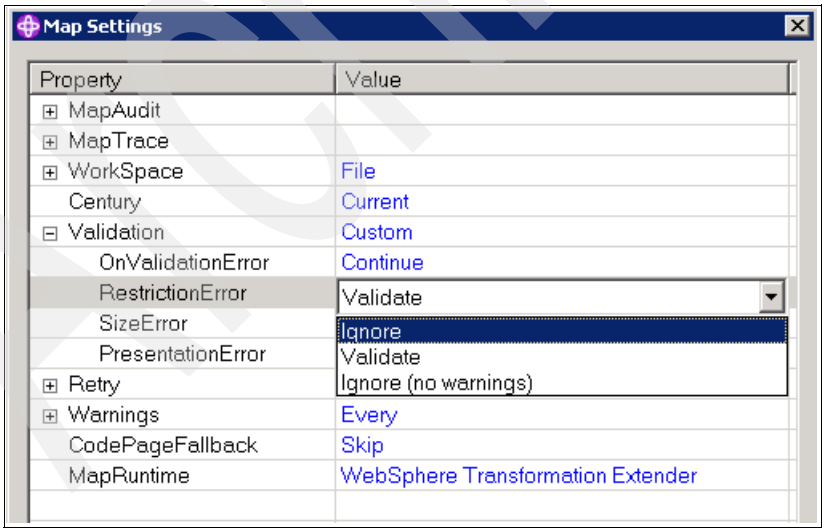

Figure 5-60 Map Settings window for restriction lists

Restriction settings can include or exclude the following restrictions:

- Value restrictions
- Character restrictions
- Range restrictions

When *Value* is selected as the Restrictions setting, the restriction list can identify valid data (include) or invalid data (exclude). Include or exclude values can be characters, numbers, or dates.

To add value restrictions to an item (Figure 5-61):

1. In the Type Tree window, double-click the item.
2. In the Include column, enter the values.
3. Optional: In the Description column, enter the descriptions of each restriction (for documentation purposes).

> **Tip:** You enter the values on the rule bar. Click the column you want to add or edit, make the change, and press Enter to save the entry.

4. Save the restriction list when completed by either pressing Ctrl+S or selecting **File** → **Save**.

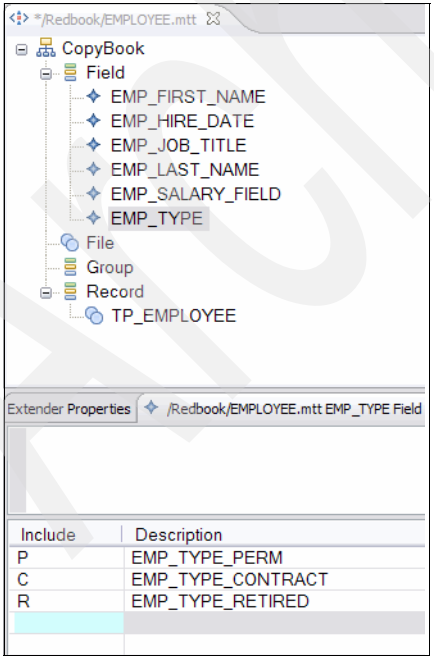

Figure 5-61 Value restrictions

When character text has character restrictions, and the rule is *Include*, the item window has two columns: one for the character list of the first character (Include First) and one for the character list of the characters that might follow (Include After).

To include character restrictions:

1. Double-click the item in the type tree.
2. Enter the values in restriction columns:
   - Include First, the character list of valid first characters
   - Include After, the character list of valid characters that might follow the first character

To specify a common range of characters, enter the reserved words in the item window cell:

```
<LETTER/>
<LETTERUPPER/>
<LETTERLOWER/>
<DIGIT/>
```

In the example in Figure 5-62, the restriction settings for Catalog# indicate that it must start with an uppercase alphabetic character (as identified by <LETTERUPPER/>) and can be followed by uppercase or lowercase alphanumeric characters or numbers (as identified by <LETTER/> and <DIGIT/>). Special characters, such as a dash (-) or question mark (?), are invalid unless they were explicitly listed in the Include After list.

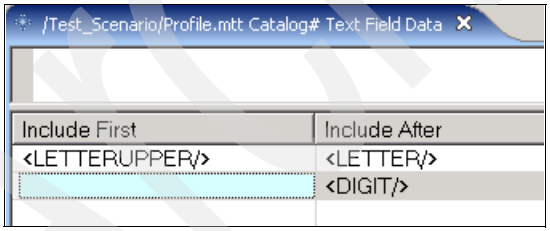

Figure 5-62 Character restrictions for Include

Exclude is list of character substrings to be excluded. When the rule is Exclude, the item window has two columns (Figure 5-63 on page 230): one for the character list to exclude (Exclude) and one for the reference replacement string (Reference String). *Reference String* is a character string that replaces the excluded character substring on output.

Exclude	Reference String
>	&gt;
<	&lt;
&	&amp;

Figure 5-63 Character restrictions Exclude

When the Restrictions property is set to Range and the Rule property is set to Include (as in Figure 5-64), the minimum and maximum specify *valid* ranges (Figure 5-65). When the Restrictions property is set to Range and the Rule property set to Exclude, the minimum and maximum specify *invalid* ranges. Include or exclude value range restrictions are valid for numbers and dates.

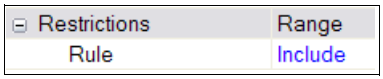

Figure 5-64 Range restrictions

Include Minimum	Include Maximum	Description
15000	95000	Salary range available

Figure 5-65 Range restrictions settings

Component rules

In the Component view, you can enter component rules to help validate the data. A *component rule* is an expression or statement about one or more components in a component list. A component rule evaluates to TRUE or FALSE. The component rule indicates what must be true for that component to be valid.

Any reference to a component in its component rule (the rule next to that component in a component list) can be replaced by the dollar sign ($). You can think of it this way: A $ always means “whatever is displayed to the left of the rule or in the Component column.”

Tip: The shorthand notation (for example, by using the $ sign) can also be used in mapping rules in the Map Editor.

Component rules are created in the rule bar (Figure 5-66 on page 231). A component rule must reference other components by the full component name. You press and hold the Alt key to drag components to the rule bar (without erasing what is already entered). Component rules can also use functions and comments, which is similar for map rules. Remember to press Enter to save the rule bar changes.

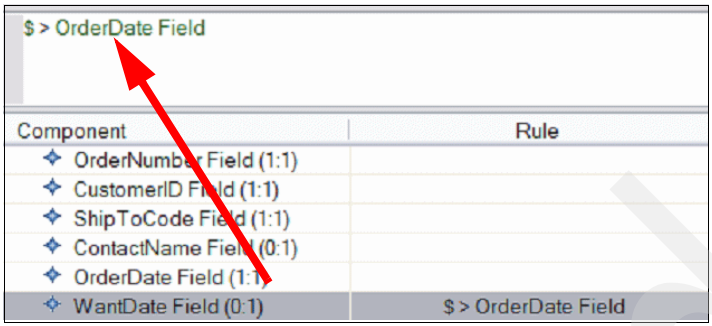

Figure 5-66 Component rule

Tip: Use WHEN(PRESENT…) if the component rule should only be evaluated when data is present.

Creating component rules

A component rule must reference other components by the full component name. In the previous example, when the component rule references the order date, it uses the full component name (ORDER_DATE Field).

The best way to do that is to press and hold down the Alt key and drag the name of the component to the rule bar. Remember to press Enter to save the changes.

The component rules can use most of the prebuilt functions of WebSphere Transformation Extender. Right-click the component rule and select **Insert Functions**. In the Functions view (Figure 5-67), drag the selected function to the Component Rule view.

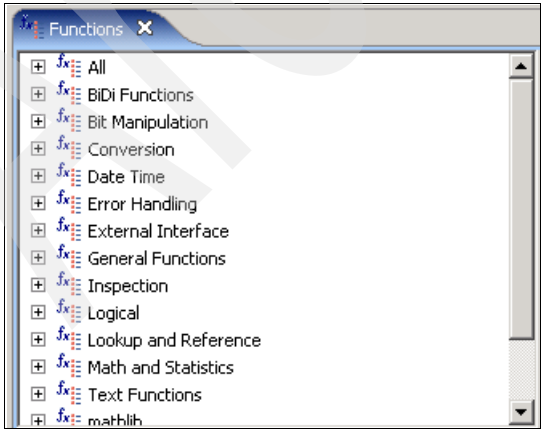

Figure 5-67 Functions view

Figure 5-68 shows an example.

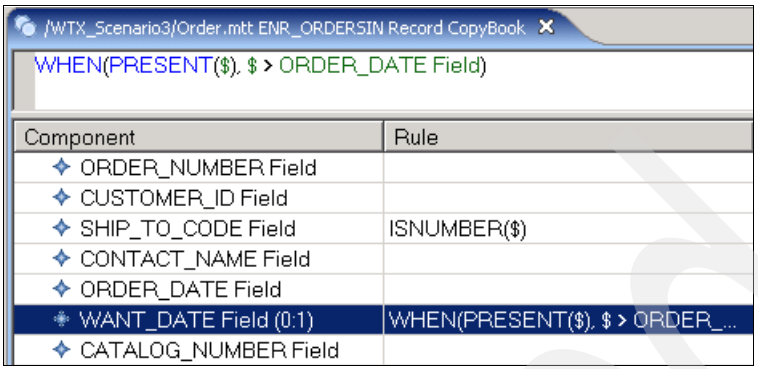

Figure 5-68 Component rules

In this example, we have two component rules:

- In SHIP_TO_CODE, the rule is ISNUMBER($), meaning that WebSphere Transformation Extender will check if the value for SHIP_TO_CODE is numeric.
- In WANT_DATE, the rule is WHEN(PRESENT($), $ > ORDER_DATE Field). This means that WebSphere Transformation Extender will check if the WANT_DATE has any value. If it does, then this value must be greater than the date in ORDER_DATE. ORDER_DATE can be used in the component rule for WANT_DATE because it is displayed earlier in the component list.

You can add comments to a component rule. A comment has the following characteristics:

- It begins with /* and ends with */.
- It is displayed anywhere in a rule.
- It cannot be placed within the object name. Placing a comment within an object name is flagged as an error by the analyzer.

Restriction list or component rule? Sometimes it is difficult to decide what to use to do validation: a restriction list, a component rule, or a mapping rule. Follow these rules of thumb:

- If you are trying to specify logic that affects validation, use a restriction list or a component rule. For example, if you have a rule that, in order to be valid a field in a group must be greater than 0, this is best entered as a component rule and not as a restriction list. Remember that restriction lists do an exact literal comparison between the data object and the entries in the restriction list. Entering >0 as a restriction compares the value in the field (for example, 35) to the value >0. That is 35 is not equal to >0.
- Use a restriction list if the validation to be added centers around the fact that a particular item in your data can only be one of a specific set of values in order to be valid.
- Use a mapping rule if the logic pertains to the output data and seems to express how an output object is determined or calculated.
- You can enter component rules in type trees that will be used for output data. Component rules are only evaluated during data validation. Output data is only evaluated when appropriate audit settings are created in the source map.

Partitioning

Partitioning can also be valuable to build additional logic into the definition of the data or to simplify the rules that will be required to map the input data to the output data. To do this by changing the value of the property Partitioned to *Yes* in the Properties view.

Example 5-4 represents a file of unordered data. The file contains three different types of transactions: invoices, orders, and remittances. Each of these transactions has a different definition based on the type of information that it represents.

Example 5-4 Unordered data file

```
BGI-13100,REM,931104,19970424…
AXR-10930,INV,003X-114,19970422…
PVY-19496,ORD,PO-104-1499,19970425…
BGI-13100,ORD,PO-182-2587,19970425…
AXR-10930,INV,003X-114,19970422…
PVY-19496,REM,931104,19970424…M
```

In this example, the second field in each record identifies the type of data in the record:

- REM indicates that the record is for a remittance.
- INV indicates that the record is for an invoice.
- ORD indicates that the record is for an order.

Figure 5-69 shows a partial definition of this file. It shows the Transaction object partitioned into the three different transaction types. As each transaction is validated, the Transformation Extender engine determines the partition to which it belongs and validates it according to that partition's definition.

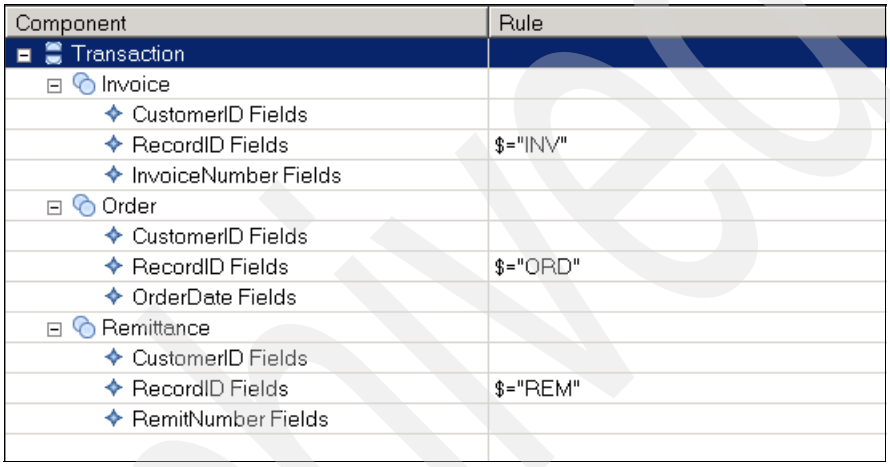

Figure 5-69 Example of partitioning

Type tree differences

A feature in Design Studio allows you to compare two type trees. This feature is particularly useful for version control and debugging when it is necessary to track changes made to a type tree.

The differences are displayed by navigating through the resulting comparison windows. The results show items that exist in one source but are missing from another source. They also show differences in formats, setting, and so on. Figure 5-72 on page 237 shows an example of the results of comparing two type trees.

Two type trees are considered different under the following circumstances:

- Any of the types are different.
- Any of the types that exist in one type tree do not exist in the second type tree.
- The order of the subtypes within a subtype is different.

A type is different under the following circumstances:

- Any of the properties are different.
- Any of the components are different.
- Any of the restrictions are different.

A restriction is different under the following circumstances:

- The restriction value is different.
- The description is different.

To compare type trees:

1. From the Design Studio menu, select **File** → **Type Tree Differences**.
2. In the Select First File window (Figure 5-70), select the first type tree (.mtt) file and click **Open**.

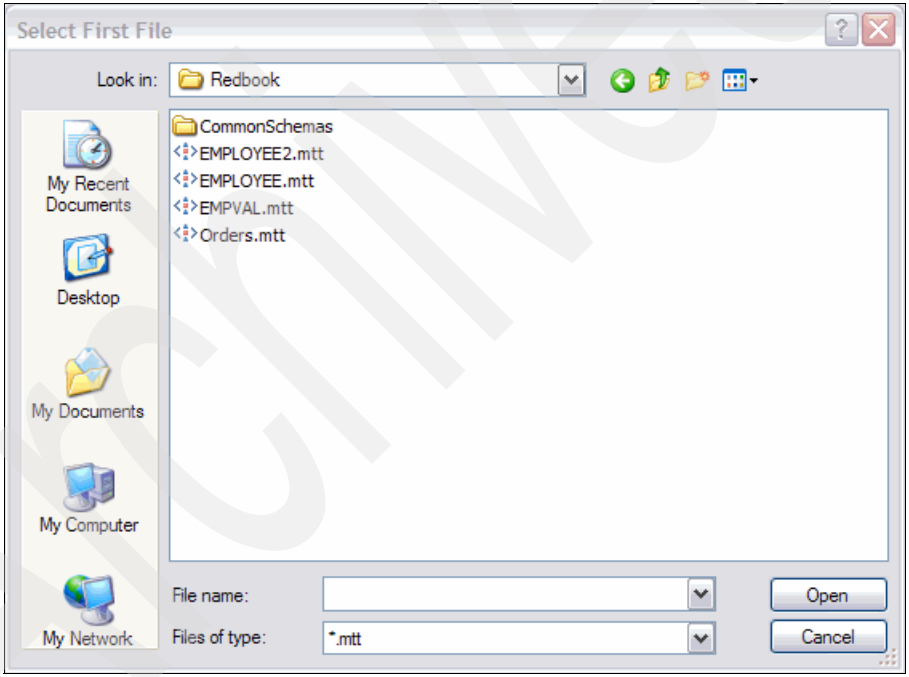

Figure 5-70 Selecting the first type tree

3. In the Select Second File window (Figure 5-71), select the second type tree (.mtt) file and click **Open**.

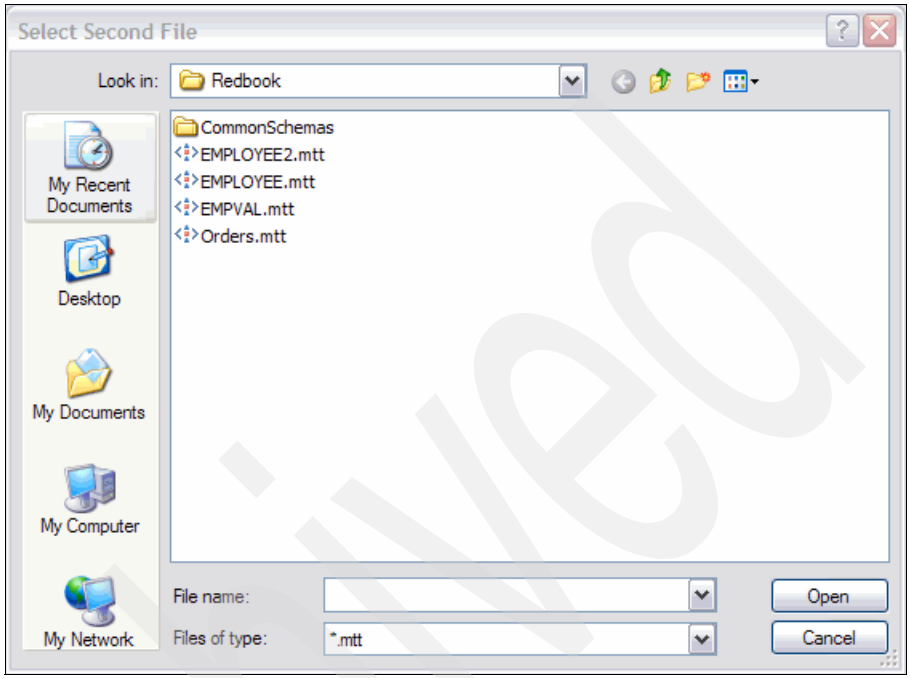

Figure 5-71 Selecting the second type tree

The Type Tree Differences window opens (Figure 5-72 on page 237), showing the two trees side by side. The top portion contains the tree sources, and the lower portion indicates the type of differences. Objects that are determined to have some difference are highlighted in the color red, while objects that do not exist in one of the source files are highlighted in blue.

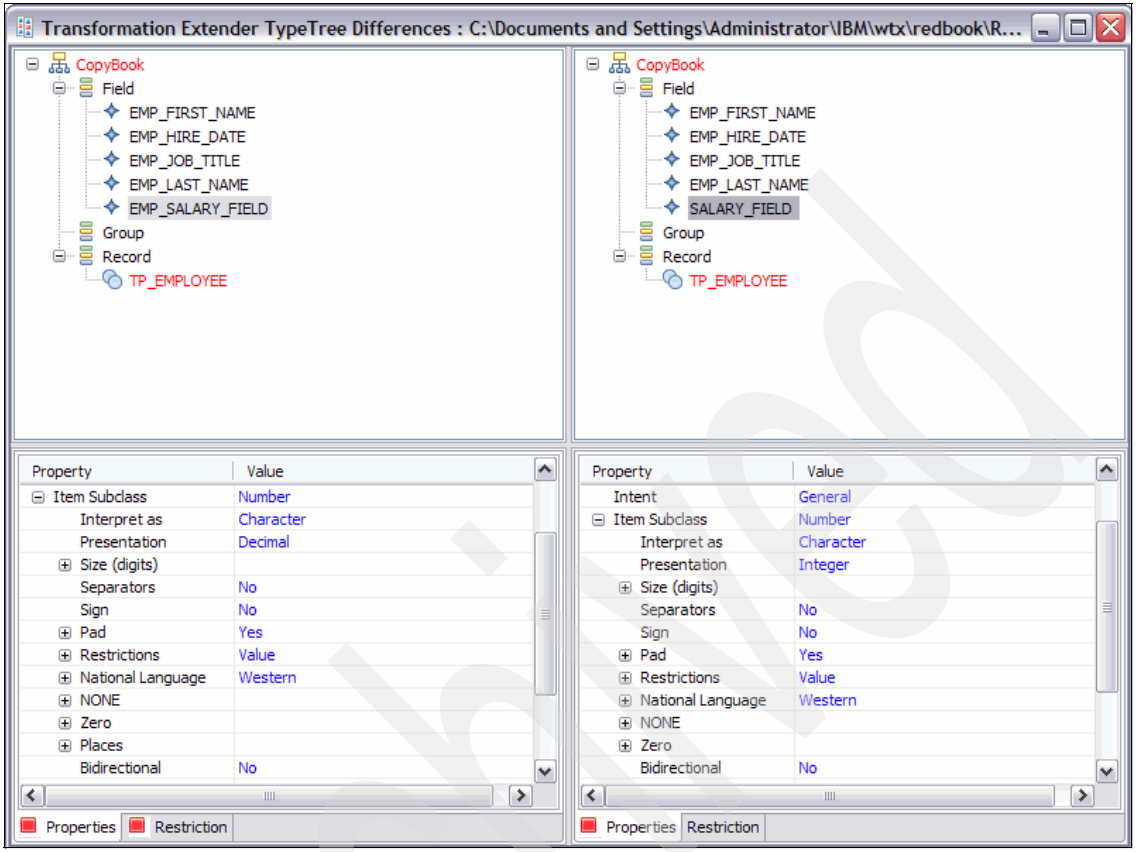

Figure 5-72 Type tree differences

5.3.3 Analyzing type trees

Use the Type Tree Analyzer to analyze your type definitions. The Type Tree Analyzer analyzes type definitions and ensures internal consistency. For example, if you defined a group as fixed, but defined one of its items as having a variable size, the analyzer issues a warning message indicating the conflict.

The analyzer checks your data definitions for logical consistency. It does not compare your definitions and actual data. The resulting analyzer messages indicate whether your type tree definitions are acceptable and not whether they match your data.

The analysis indicates whether there is something in your definition that might prevent a correct mapping of your data.

To analyze a type tree, with the type tree open, select **Tree** → **Analyze** and then select the type of analysis you want to run, for example Structure and Logic, Logic Only, or Structure Only (Figure 5-73).

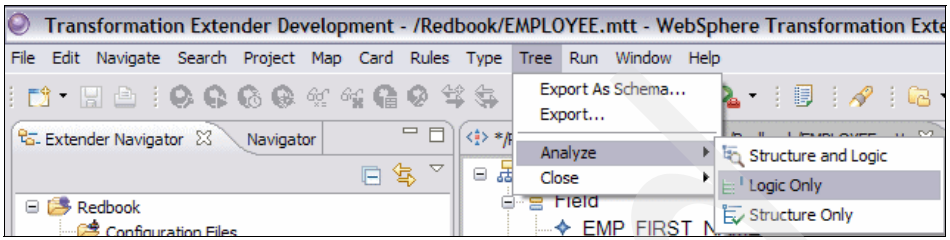

Figure 5-73 Type Tree Analyzer

After the analysis runs, the Analysis Results view opens. This view shows the name of the type tree, the type of analysis, and the status of the process. When the analysis process complete, the number of errors and warnings are displayed.

Type tree analysis results

The analysis results include one or more messages if problems are encountered during type tree analysis. Errors indicate that the tree violates a syntactical rule, such as having a component that is displayed twice in a component list. All errors must be resolved by fixing the problem.

Some warnings indicate that Design Studio has taken corrective action, such as removing a restriction list when an object that was an item has been changed to be a group. Other warnings indicate a possible inconsistency in the tree that might cause maps that use the tree to produce unpredictable results, such as being unable to distinguish one object from another that might follow it in the data. Warnings usually indicate an inconsistency that should be resolved.

The type tree analysis includes the following result messages:

- Error number
  - "L" for error in logical analysis
  - "S" for error in structural analysis
- Text message that describes the error
- (error) or (warning)

Figure 5-74 shows an example of a type tree analysis result. In this example, the types in error are File Data and Header Record Data. The results also show warnings in Detail Record Data.

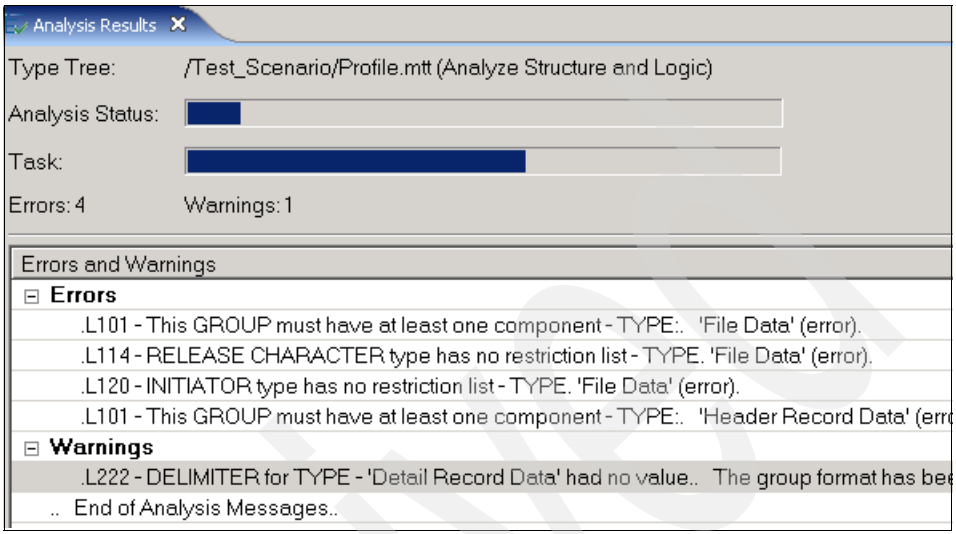

Figure 5-74 Analysis results

For a complete description of all errors and warnings together with hints on how to resolve them, see the WebSphere Transformation Extender Online Library.

If you attempt to build a map that references type trees that have not been analyzed as error- and warning-free, a message indicates that unpredictable results might occur, as shown in the Warning window in Figure 5-75.

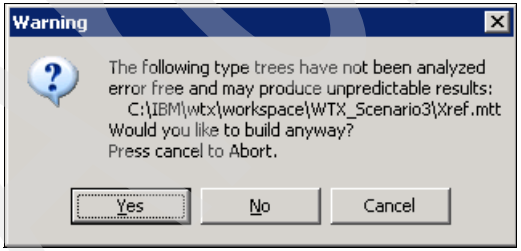

Figure 5-75 Map build warning message

Important: Sometimes you see this warning message after analyzing the type tree. This message might occur because you forgot the save the type tree after you analyzed it. Make sure that you *always* save the type tree after analyzing it successfully, because the analysis results are stored with the type tree.

5.3.4 Mapping

After defining data objects and their properties in the Type Tree Editor, define the map in the Map Editor where you specify the source of the input and the target of the output. Sources and targets are specified with map cards. Each map can have none or more input cards and one or more output cards.

Remember that there are two types of maps:

- Executable map

  An executable map is responsible for the totality of your inputs and outputs. The sources and targets of an executable map are entire files, database tables, messages, applications, and so on. Think of an executable map as the main, top-level map. Executable maps are compiled and run.

- Functional map

  A functional map is referenced by another map through a map rule. A functional map maps just a portion of the entire data. Data sources and targets are not specified in the cards of functional maps. Functional maps are not compiled and run.

Figure 5-76 illustrates the methodology for building maps in the Map Editor.

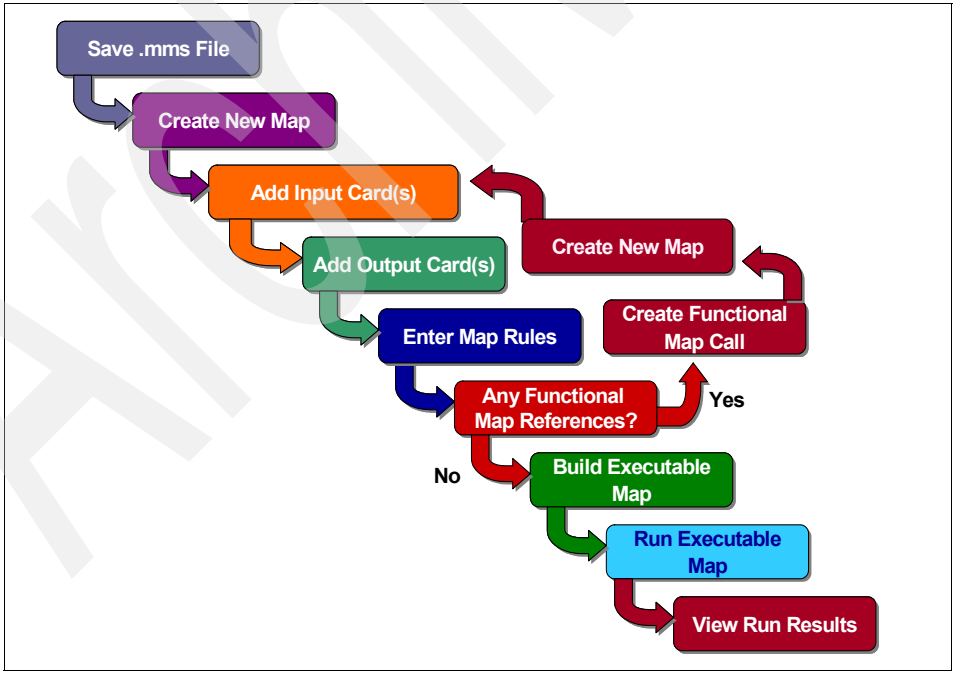

Figure 5-76 Map design methodology

To create maps in the Transformation Extender Development perspective:

1. Create a map source file with the .mms extension.
2. Create the required executable map or maps in the map source file (.mms).
3. Create an input card that represents your input data.
4. Create an output card that represents your desired output.
5. In the output card, enter the map rules, specifying the logic that is necessary to transform the input data to the desired output data.
6. Build the map, which creates a compiled map file (.mmc).
7. Run the map.

A map defines how to generate a data object of a certain type or multiple independent data objects, each of a certain type. A map contains input and output cards that transform data content from source formats to destination formats according to rules in the map.

Figure 5-77 shows each important part of the Map Editor that we describe.

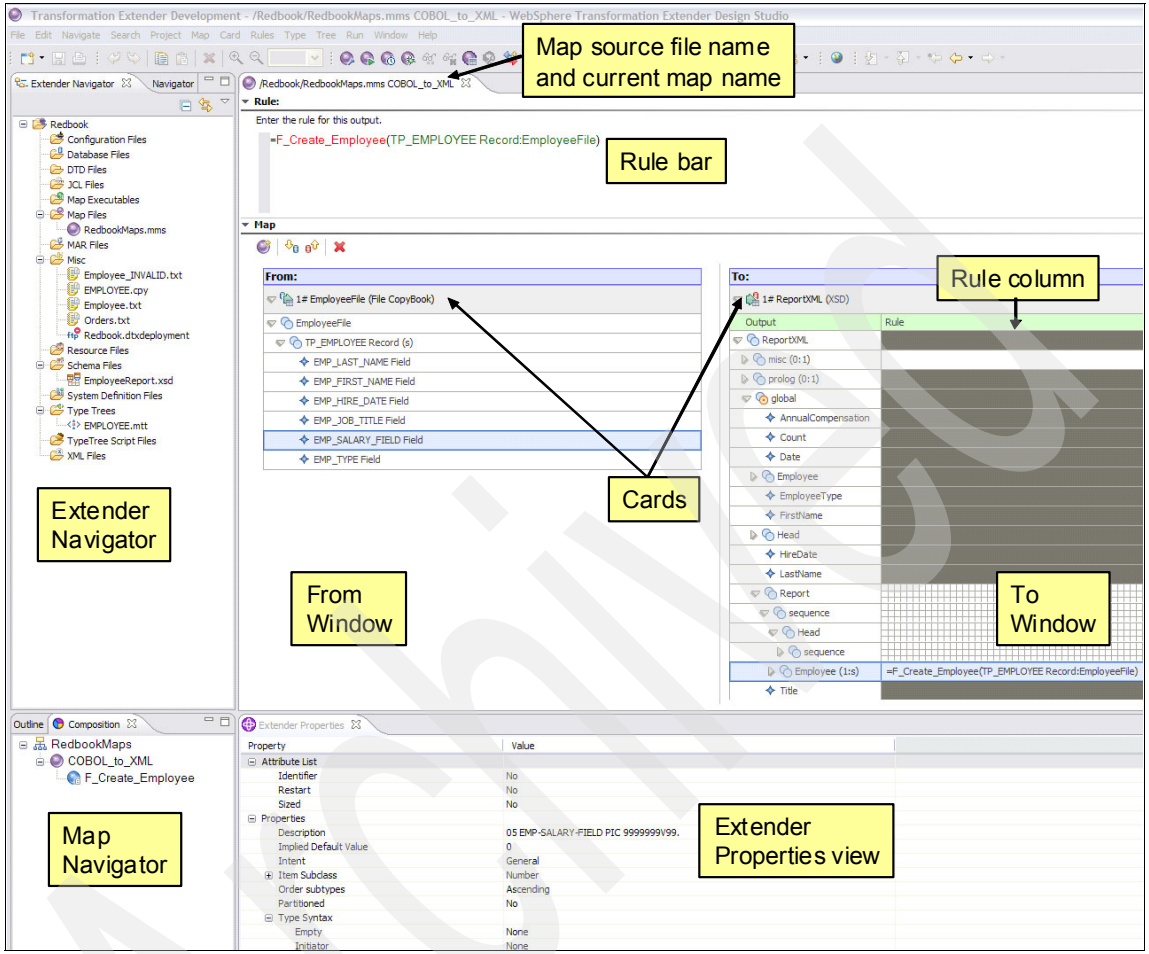

Figure 5-77 Map Editor components

Map source file

A *map source file* is used to store maps that are related to one another. To create a new map source file:

1. Use either of the following methods:
   - Select **File** → **New** → **Map Source** from the Design Studio menu.
   - Right-click the **Map Files** folder in your project and select **New** → **Map Source** (Figure 5-78 on page 243).

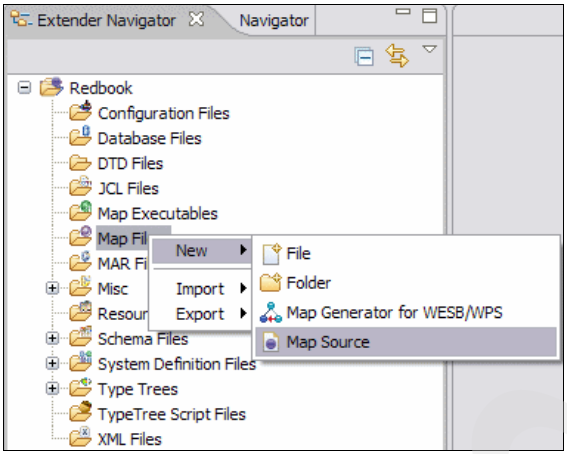

Figure 5-78 Selecting a new map source from the Maps files folder

2. In the Map Source window (Figure 5-79), provide a map source name. Select the Project folder for the map source file and click **Finish**.

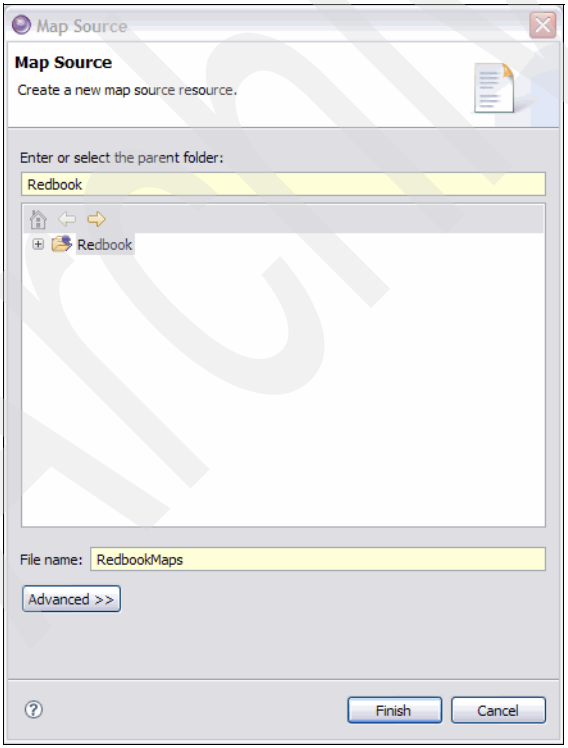

Figure 5-79 Creating a new map source

A map source file has the .mms file extension. A map source file maintains maps in source format. Map source files contain maps and functional maps. After a map source file is created, the map source file name and icon are displayed in the Composition and Outline views (Figure 5-80).

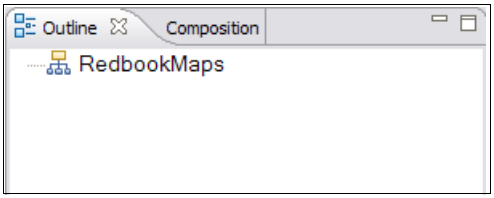

Figure 5-80 Maps Outline view

Composition view

The Composition view shows a map source file and the maps that they contain. It shows map relationships, arranged in a compositional hierarchy, with functional maps listed under the map that calls it. New maps are displayed only after at least one card is defined.

Outline view

The Outline view shows map definitions in alphabetic order. By expanding a map in the Outline view, you can see its input cards and output cards, as well as an Organizer list that you can use to display optional files that are associated with the map (such as a trace file). You can open any map in the Outline view or the Composition view by double-clicking it.

Note: All maps are shown at the same level on the Outline view, regardless of whether they are an executable map or functional map.

Maps

One or more maps are contained in map source files. To create a map:

1. In the Outline view, select the desired map source file.
2. From the Design Studio menu, select **Map** → **New**, or right-click the map source file in the Outline view and select **New** (Figure 5-81).

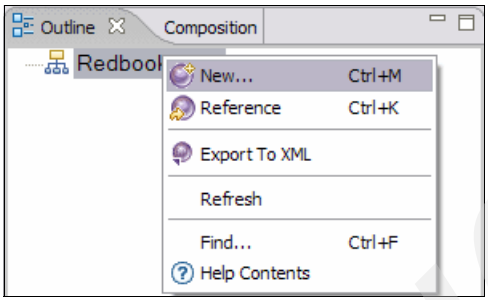

Figure 5-81 Map source pop-up menu

A new map is listed in the Outline view (Figure 5-82). You can immediately enter the name of the map. If you need to rename the map, from the Design Studio menu, select **Map** → **Rename** and enter the new name in the Map name field.

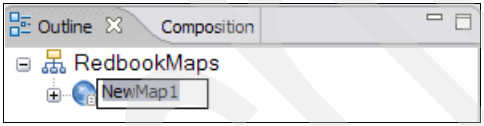

Figure 5-82 New map in the Outline view of the map source

Note: The Composition view displays a *map* only after an input or output card has been added to the map.

After you create the map, a blank map opens in the Map Editor (Figure 5-83). The map contains a From window and a To window, each of which are waiting for inputs and outputs to be defined.

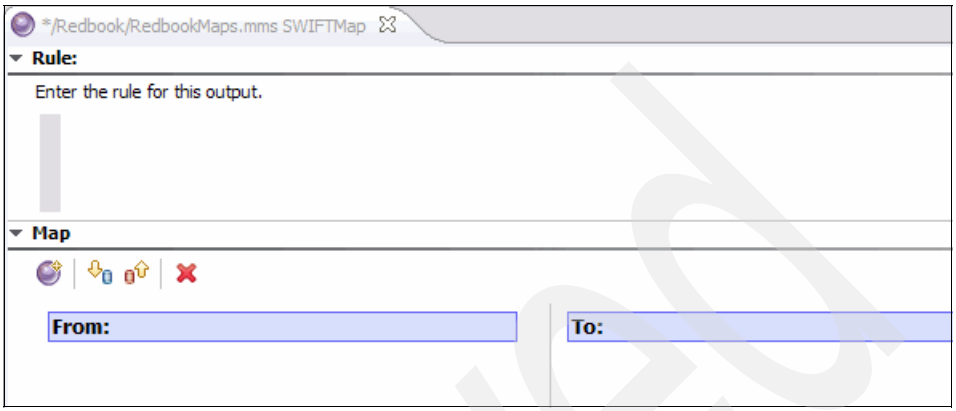

Figure 5-83 Map Editor main view

Cards

The From window of the Map Editor represents the input data or the data from which you are mapping. The From window contains no or more map cards. Map cards in the From window of the Map Editor represent source data and are called *input cards*. The From window contains an input card for each data object from which you are mapping.

The cards in the From window show the composition of the data defined in the Type Tree Editor. An input card contains the complete definition of an input for the map including information such as source identification, retrieval specifics, and the behavior that should occur during processing.

The To window of the Map Editor represents the output data to which you are mapping. The To window contains one or more map cards. Map cards in the To window of the Map Editor represent target data and are called *output cards*. The To window contains an output card for each data object to which you are mapping.

An output card contains the complete definition of an output for the map including information such as target identification, destination specifics, and the behavior that should occur during processing.

A card can represent an object in a row, a row in a record, the entire record, or the entire file. The maximum number of input cards in a map is 360, and the maximum number of output cards in a map is 360. Cards are numbered sequentially and determine the order in which the cards are run.

The create card function relates directly to which window is active or which card type (input cards or output cards) in the Outline view is selected when the new card is created. To create an input card, the From window or the Input Cards element in the Outline view must be active. To create an output card, the To window or the Output Cards element in the Outline view must be active.

To create an input card:

1. Under the map in the Outline view, select the **Input Cards** element.
2. From the Design Studio menu, select **Card** → **New**, or right-click **Input Cards** and select **New** (Figure 5-84).

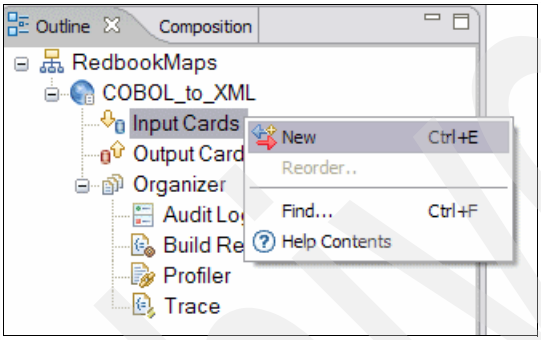

Figure 5-84 Creating an input card

3. In the New Input Card window (Figure 5-85 on page 248):
   a. Define the card by entering the appropriate information for each setting. At a minimum, define the following values:
      - The CardName that describes the input
      - The TypeTree that contains the input data definition
      - The Type within the type tree that defines the input data
      - The Source resource adapter
      - Any adapter specific settings
   b. Click **OK**.

   The card is created in the currently selected map.

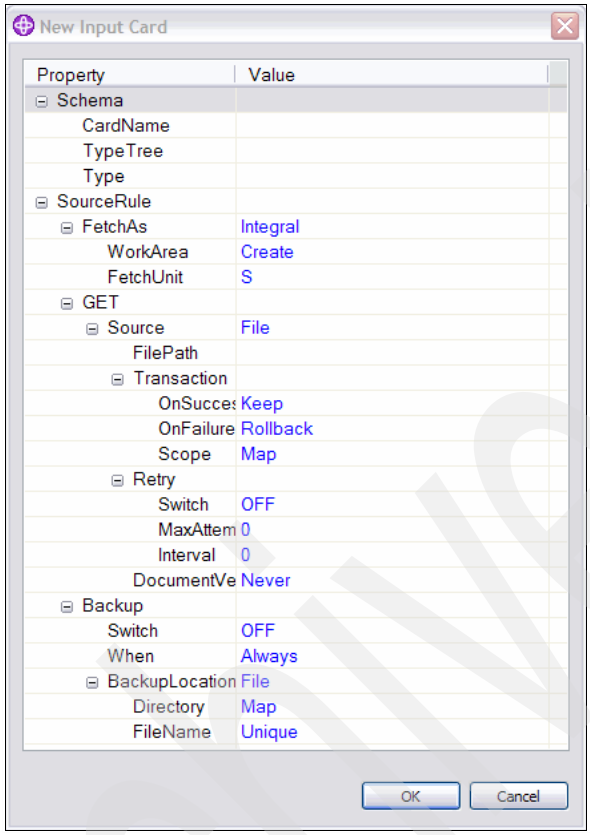

Figure 5-85 New Input Card window

To create an output card:

1. Under the map in the Outline view, right-click the **Output Cards** element and select **New** (Figure 5-86). Alternatively, from the Design Studio menu, select **Card** → **New**.

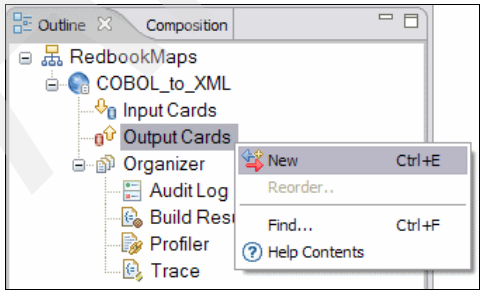

Figure 5-86 Creating an output card

2. In the New Output Card window (Figure 5-87):

   a. Define the card by entering the appropriate information for each setting. At a minimum, define the following settings:

      - The CardName that describes the output
      - The TypeTree that contains the output data definition
      - The Type within the type tree that defines the output data
      - The Target resource adapter
      - Any adapter specific settings

   b. Click **OK**.

   The card is created in the currently selected map.

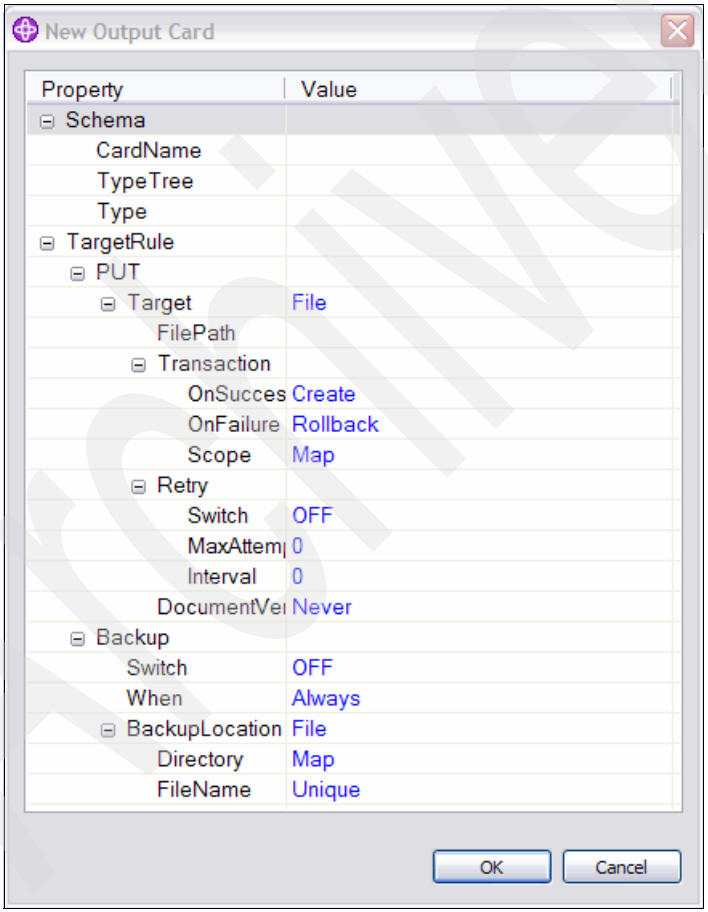

Figure 5-87 New Output Card window

Another way to create input and output cards is to click the icons for add input card or add output card (circled in Figure 5-88) under the Map section heading.

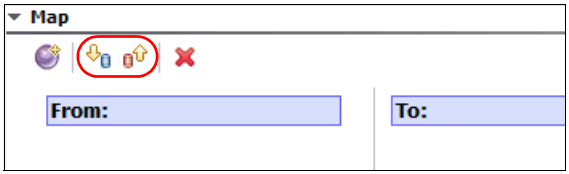

Figure 5-88 New input and output card icons

Tips:

- Precise naming of the card is practical and useful. Provide a name that provides a description of its contents so you know the contents by just looking at the card name. When mapping to and from multiple cards, it helps to instantly recognize that you are touching the correct card. Name the card the same name as the data object that it represents.
- The default is to show only one card in each window in the Map Editor. To change the default so that you can see multiple input and output cards, change the preferences under **Window** → **Preferences** → **Transformation Extender** → **Map** (Figure 5-89).

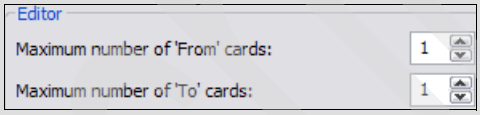

Figure 5-89 Number of cards shown option

Enabling the map executable to be built: To enable the map executable to be built (meaning that the option is available to you), on every input card, you must define an available “Source” property within the GET structure. In addition, on every output card, you must define an available “Target” within the PUT structure. Use the SINK adapter to generate no physical output.

The cards in the To window have two columns: the Output column and the Rule column. The *Output column* lists the name of each output data object. The Output column shows the compositional hierarchy of the output data objects. The *Rule column* contains the mapping rules that provide the logic for building each object in the output.

Map rules

Map rules are used in output cards of a map to define how output data is built. Map rules are required for output data generation. If there is no map rule, there is no output data. When no output is desired for a given data object, a map rule of =NONE is required. Map rules can be as simple or as complex as needed to build a particular output data object.

A map rule can have the following functions:

- Map an input object to an output object
- Extract input records based on specific criteria
- Count a conditional number of records
- Map input records based on specific qualifier to a specific output record
- Convert an object in input data from one value to another value
- Perform conditional logic
- Run another map to produce an output

Map rules are entered onto the map rule bar. A map is not complete until each rule cell has a map rule, or is displayed as disabled. Empty rule cells generate a build error when the map is built.

Color-coding of map rules provides visual ease of use in formulating and understanding rules. When color-coding is enabled, syntax errors in map rules are underlined. Map rules are validated only when that map rule is changed.

Tip: To enable color-coding of map rules, select **Window** → **Preferences** → **Transformation Extender** → **Type Tree** → **Group Window** and, under Rule Color Specifications, set the colors.

A map rule begins with an equal sign (=), followed by an expression that evaluates to data. The maximum length of a map rule is 32K characters (including non-printable characters). An *expression* is any valid combination of literals, data object names, operators, functions, and map names. The expression in a map rule generates the desired data.

A map rule for a data object must evaluate to a data object with the same subclass. To view the type properties of a data object in a card, right-click the type in the input or output card and select **Show in Properties**. If the Extender Properties view is already shown, select the map rule to update the Extender Properties view to show the properties of the object.

Tip: To review or alter the type tree that a card is using, click any object on the card, right-click, and select **Open Type Tree** (Figure 5-90). (You do not have access to this option if you click the card [gray box] itself.) By doing so, you ensure you are opening the correct type tree that the map is using.

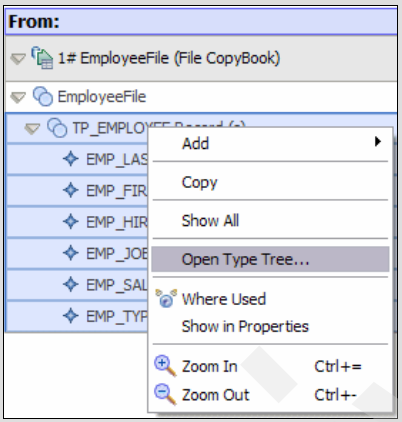

Figure 5-90 Selecting Open Tree Type

If you alter the type tree, analyze the type tree, save the type tree, and click **Yes** when asked to update the open map with the changed type tree.

Basic mapping involves simply dragging from the input source to the output target. However, many map rules require more than a simple equality, such as a function to alter or enhance the output.

You enter map rules by using any of the following methods:

- Typing
- Dragging a data object from an input card
- Copying and pasting a data object from an input card
- Dragging a map name from the Extender Navigator view
- Inserting a function
- Dragging an item restriction from the Extender Properties view into the rule bar

To enter an object name in a rule, drag the object from the input card to the rule. It is easier than typing the name and reduces the possibility of entry errors. After completing the map rule, press Enter (the default commit key) to save and commit the rule to the map. Pressing Enter also causes the rule to be checked and reports any syntax errors. The cursor is automatically positioned at the first syntax error. When color coding is enabled, syntax errors can be configured to display with an underline.

Figure 5-91 shows a map and the map rule for the EMP_HIRE_DATE in the map rule bar above the map.

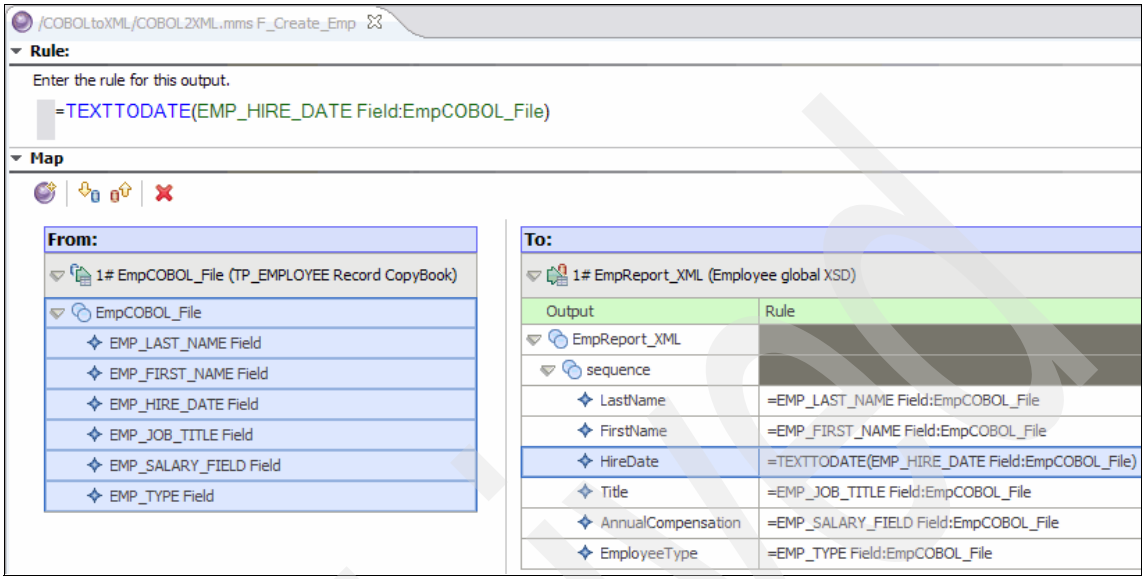

Figure 5-91 Map and map rule example

Tips:

- When entering a map rule, you must press Enter (the default commit key) to save the rule. If you press the Tab key to move off the rule bar or select another item anywhere on the map, you will lose the uncommitted rule.
- To enable or disable any confirmation message, select **Window** → **Preferences**. In the window that opens, navigate to **Transformation Extender** → **Type Tree** → **Confirmations** and **Transformation Extender** → **Map** → **Confirmations**.

Figure 5-92 shows an example of dragging an input object directly to the output object when the types are the same and it is a simple equality.

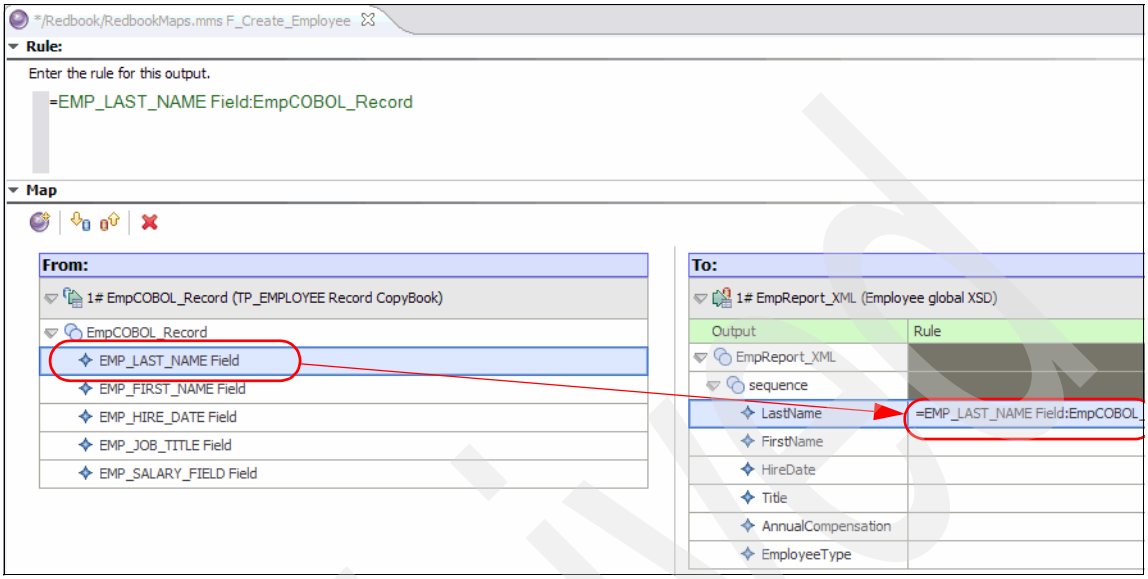

Figure 5-92 Creating a map rule

To insert a function into a map rule:

1. Select the rule cell where you want to insert the function.
2. If necessary, click in the rule bar in the specific place where you want to insert the function.
3. Right-click in the rule bar and choose **Insert Function**. (If the Function View is already open, you can drag the function to the rule bar.)

   The Functions view is displayed (see Figure 5-93).
4. If you know the function category, expand the category folder.
5. Drag the function that you want to insert it into the rule bar.
6. Insert another function if you want.

   Tip: In the Functions view, when you click a function, its description and syntax information are displayed in the right pane of the view.

7. Optional: Close the Functions view.

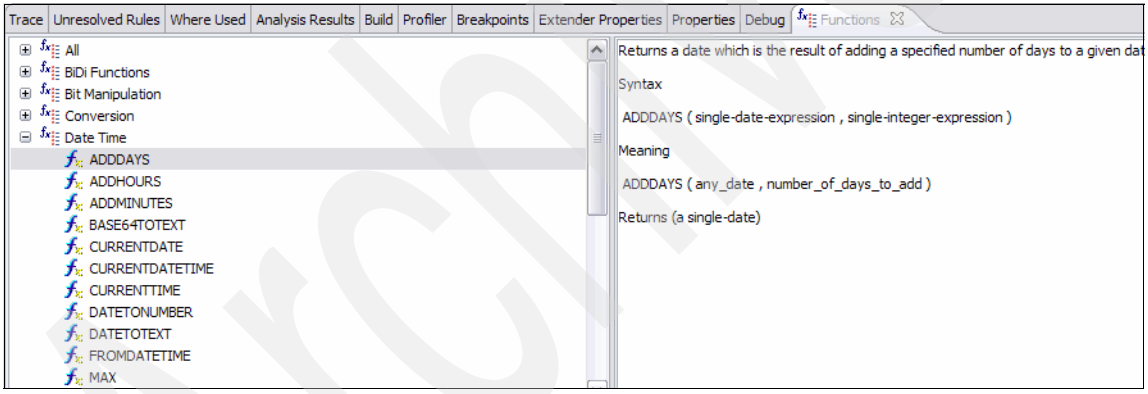

Figure 5-93 Functions View

Figure 5-94 shows example of dragging a function to the map bar and dragging an input object (into that function) to the map bar to build a map rule.

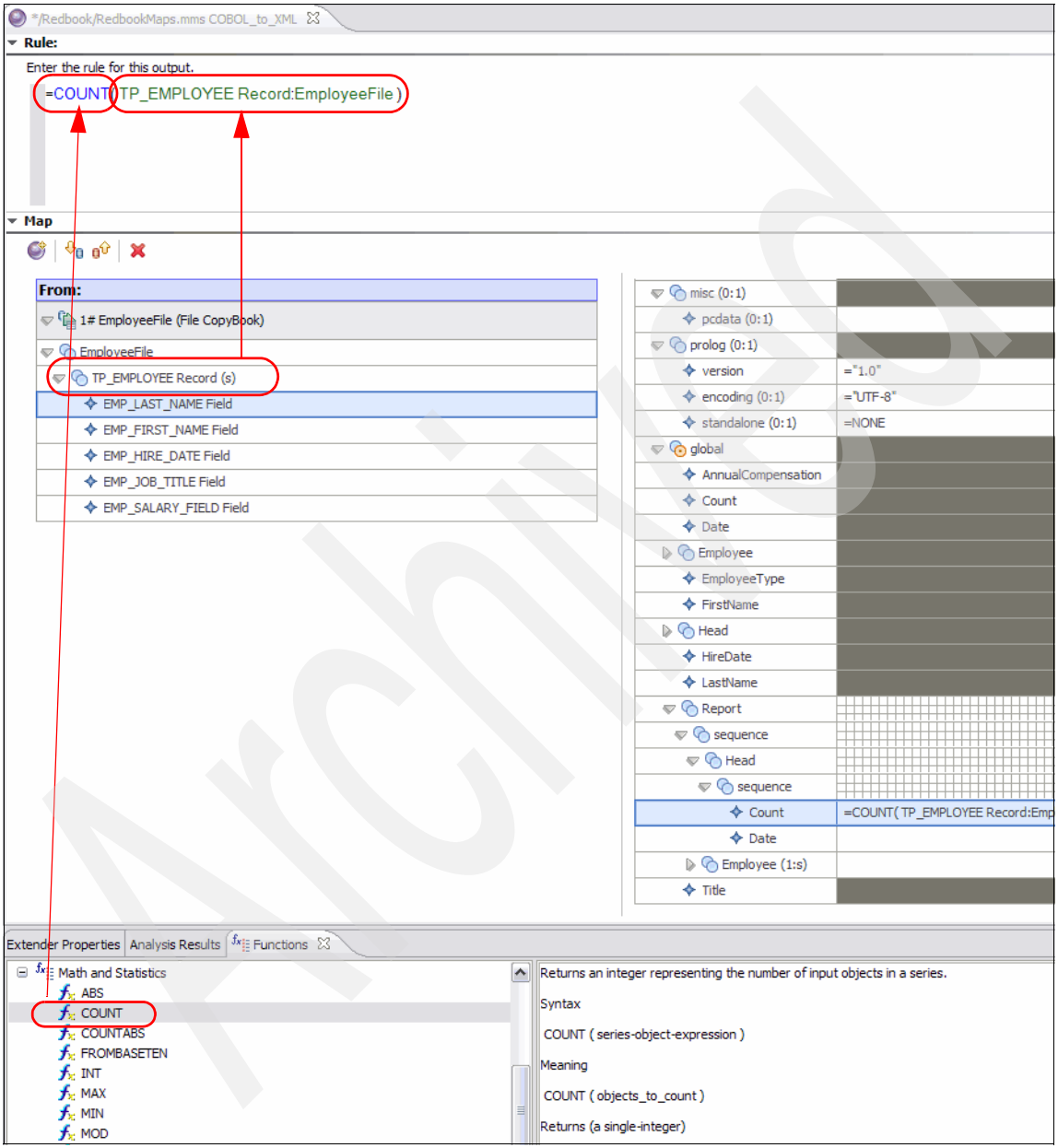

Figure 5-94 Creating a map rule by using a function

Each map rule is evaluated independently from all other rules. Each output object with a rule is created independently from the others.

Map rules are evaluated in sequential order. When a map is executed, the first rule of the first output card is evaluated. Next, the second rule of the first output card is evaluated, and so on, until the last rule of the first output card. Then, the first rule of the second output card is evaluated, and so on, to the last rule of the last output card.

To enter the object name of an output data object, drag it to the Rule bar just as you do with an input data object. Output data is built from top to bottom, beginning at the first output card.

You can map an output to another output, as long as the output you are mapping from is evaluated before the output to which you are mapping. You can reference an output in the same card or another card. You can reference a data object from another output card, as long as that output card is evaluated before the output card that contains the map rule.

If an output item or group can occur more than once, you must decide how many occurrences are to be built. If there is a known number of outputs, you can use indexing. If the number of outputs depends on a variable number of inputs, you must use a functional map, as described in “Functional maps” on page 260.

You use indexing to generate a fixed number of outputs each time the map is run. To index an output is to generate a specific number of occurrences for that output. When an item is indexed, map rules can be entered for its underlying components. If a group component is indexed, you can expand it and enter map rules at the component level.

To index an output, right-click the object and select **Add Index** (Figure 5-95).

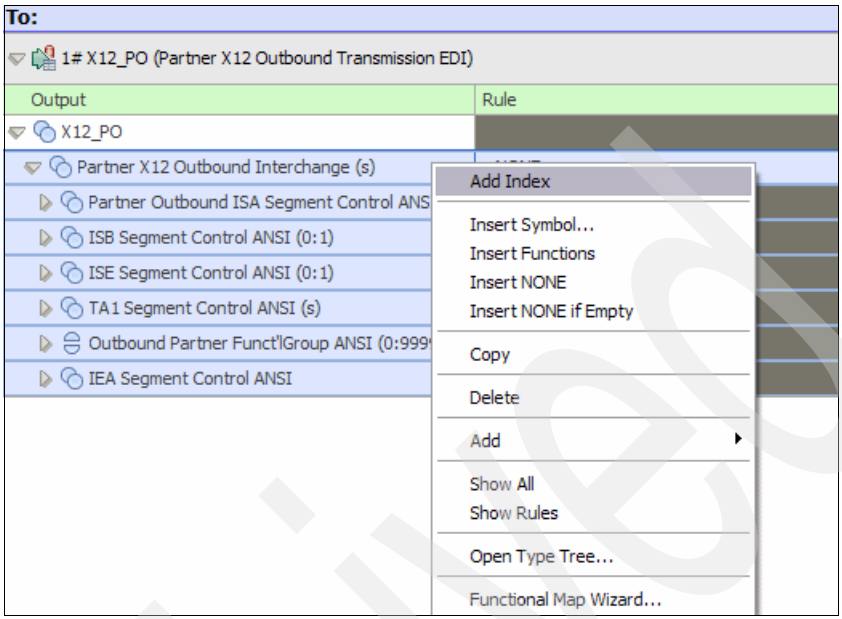

Figure 5-95 Selecting Add Index in the map rules

After adding the index, the object shows the index number (circled in Figure 5-96). You can then enter rules for that occurrence, so that in this example, you can create a single output record, knowing that you are only receiving one input object. You can add additional indexes (up to the maximum for the object as defined in the type tree), delete an index, or reorder indexes if required.

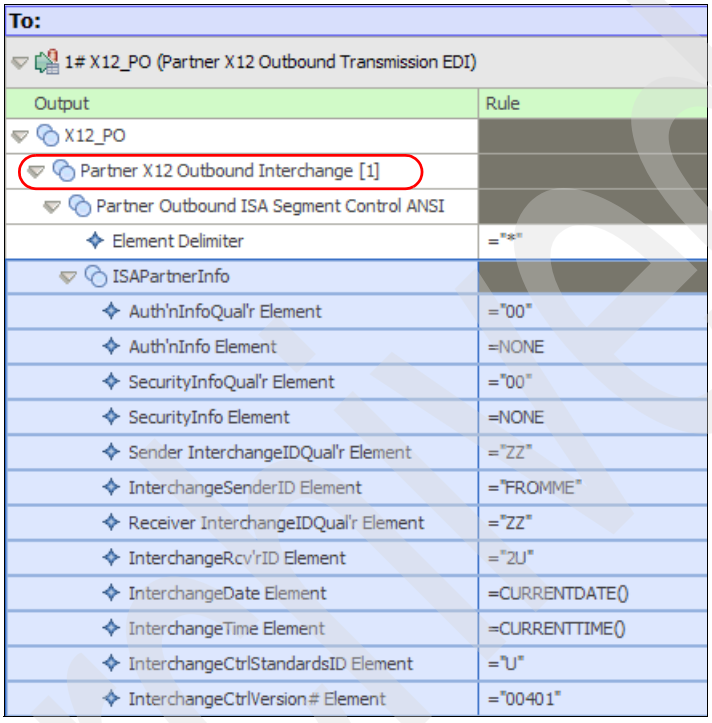

Figure 5-96 Adding an index

You can add inline comments to map rules. Comments do not affect how map rules are evaluated. A comment can be displayed anywhere in a rule, as long as it does not separate object names.

A comment is identified with the start characters /*. While the end characters */ can be present, they are not required. If the end characters are not present, everything up to the end of that rule is the comment.

The map rule shown in Figure 5-97 has two comments, which are indicated by the arrows.

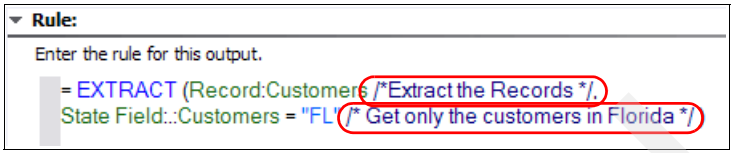

Figure 5-97 Map rule with comments

Using comments for map rules, or portions of map rules, is a convenient way to retain complex and lengthy map rules for future use without deleting them.

Rules for output objects: Remember that every output object must have a rule. To insert =NONE in every empty rule cell, select **Rules** → **Insert NONE if Empty**.

Only empty rule cells that are showing when you scroll up and down in the output card are filled. To enter =NONE in all empty cells, including the ones that are nested beneath unavailable cells, select **Rules** → **Show All** and then select **Rules** → **Insert NONE if Empty**.

Functional maps

A *functional map* is like a subroutine. It maps a portion of data at a time. A functional map is a map that is used like a function. It takes one or more input objects and generates one output object.

The use of functional maps is common. Almost every executable map that is created uses at least one functional map. With functional maps, WebSphere Transformation Extender can easily map complex data. To map a group in the input to a different group in the output, use a functional map.

For example, use a functional map to map an input row to an output row when the rows are defined differently. You also use a functional map to map from a file that contains many input rows, to generate a file of many output rows with one output row per input row. The first output row corresponds to the first input row, the second output row corresponds to the second input row, and so on.

A map defines how to generate the output data. One important factor to consider in determining when to use a functional map is the presence of an output component with a range of more than one, for example, ranges of (s) or (1:10). The number of this output object can be created based on the number of an input object.

Another important factor in determining when to use a functional map is when you want to transform the data, mapping from one or more types to a different type. Use a functional map when the number of a certain output group that you want to create is based on the occurrences of input or output data and the types are different types.

For example, you might have a functional map that maps one message to one row in a database table, or you might have a functional map that maps one header and one detail to one ItemRecord. The results of the functional map are sent directly to the output card. They are not passed back to the calling rule.

The use of a functional map entails the following basic steps:

1. Determine the need for using a functional map.
2. Determine the input argument or arguments of the functional map.
3. Enter the map rule that references the functional map.
4. Create the functional map.
5. Create the input card or cards.
6. Create the output card or cards.
7. Enter map rules in the functional map.

The Functional Map Wizard eliminates steps 5 and 6.

Map rules on the output cards of an executable map specify the functional map name. The syntax of a functional map expression is similar to the syntax of a function. Similar to a function, a functional map has one or more inputs, and one output. The inputs and the output are data objects.

The map rule for a functional map names the functional map and specifics what data to pass to the functional map. A functional map expression has the following syntax:

```
FunctionalMapName (argument1, argument2,...argumentn)
```

In this syntax, *n* is the argument number.

The input arguments of a functional map are displayed in parentheses in the functional map expression and are separated by commas. An input argument of a functional map is any valid expression that evaluates to data, including data objects, functions, and literals.

Tip: Prefix a functional map name with F_ so that when you look at the map rule it is instantly recognizable as a functional map.

The input arguments of a functional map are the objects that are necessary to create one occurrence of the output object where the map rule is. Whatever is

necessary in a functional map must be passed to it as an argument. There might be one or more input arguments. The arguments of a functional map can be thought of as *triggers*. The number of times a functional map is triggered depends on the number of occurrences of the input arguments in the data and the expressions that are used for each argument. To determine the input arguments of a functional map, ask yourself what objects are necessary to create one occurrence of the output object.

Figure 5-98 shows an example of creating the map rule to define a functional map. The functional map name, F_Create_Employee, was typed, and the TP_EMPLOYEE Record(s) object is dragged in, representing a functional map that execute once for *every* TP_EMPLOYEE Record that is in the EmployeeFile.

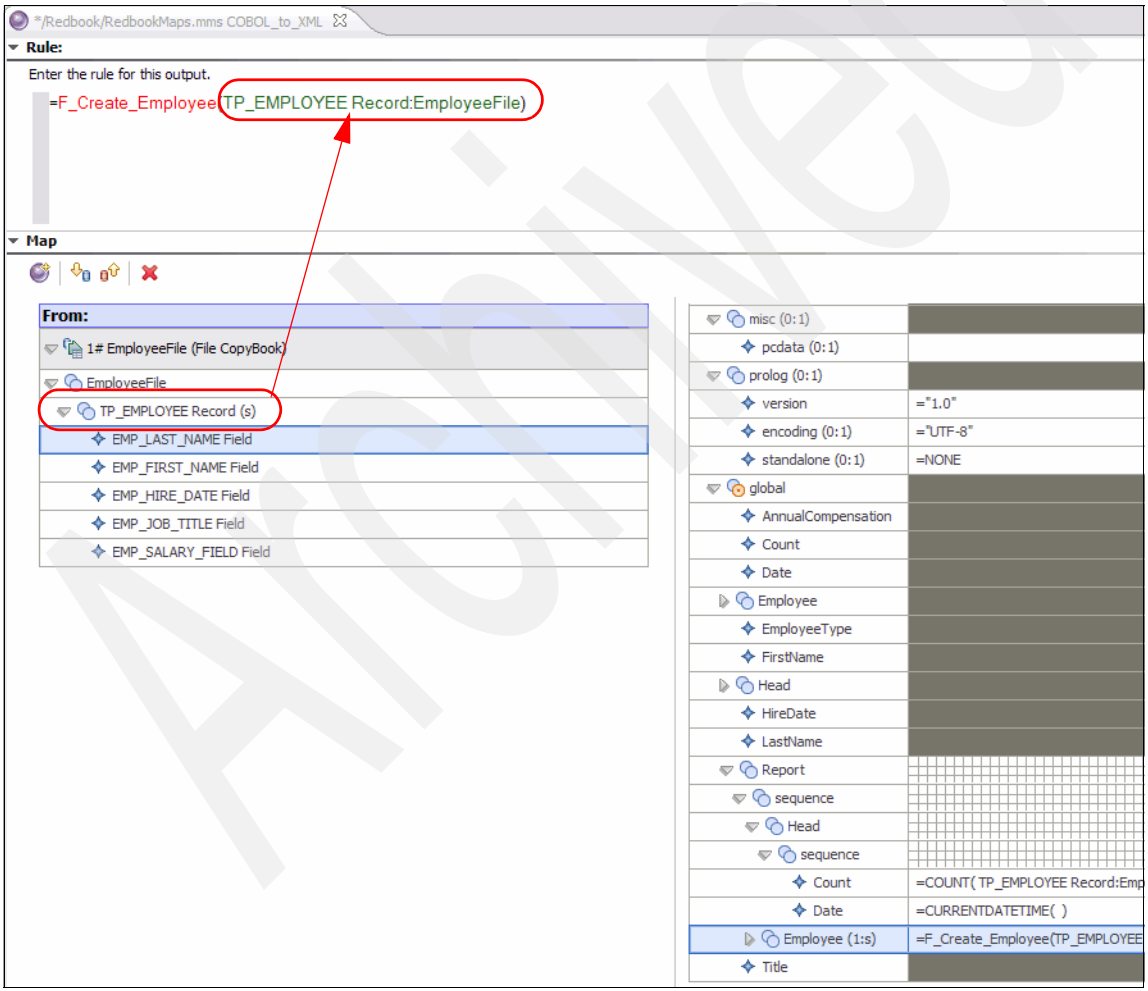

Figure 5-98 Map rule for calling a functional map

Tip: Any objects that the functional map requires *must* be passed into the functional map for it to be accessible by the functional map, or by any object down its composition path.

After you create the map rule for the functional map, use the Functional Map Wizard (or you can manually perform the steps) to create a functional map or maps based on the map rule or rules. The maps that the wizard creates contain input cards and an output card, but they do not contain map rules.

To create a functional map by using the Functional Map Wizard:

1. In the executable map, enter the map rules for the functional map.
2. Select the rule or rules for which you want to generate a functional map or maps.
3. Select **Rules** → **Functional Map Wizard** or right-click the rule and select **Functional Map Wizard** (Figure 5-99).

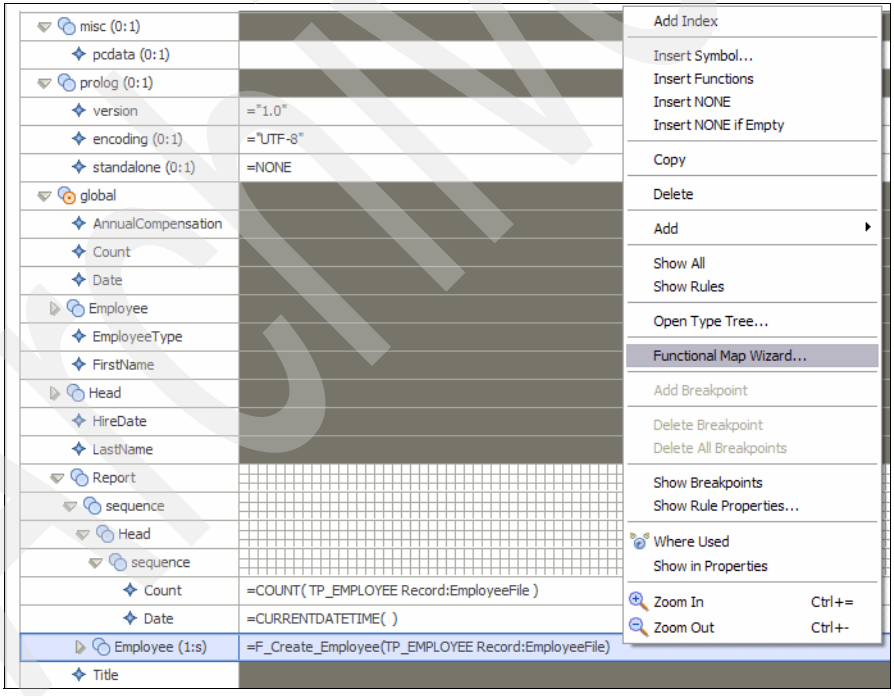

Figure 5-99 Selecting Functional Map Wizard

4. In the Functional Map Wizard (Figure 5-100), if necessary, correct problems that are related to undefined cards and maps.

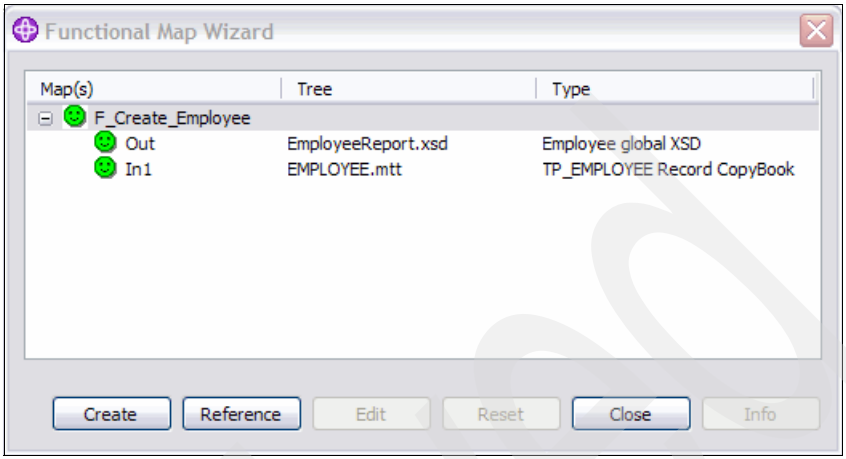

Figure 5-100 Functional Map Wizard

5. Edit the cards as required. Assign meaningful names to the cards in the functional map. Name the card the same name as the data object it represents. To edit a card, select the card and click **Edit**.

   Figure 5-101 shows an example of editing the output card shown in Figure 5-100. In this example, we change the name from Out to something more meaningful that represents the data, `EmpReport_XML`.

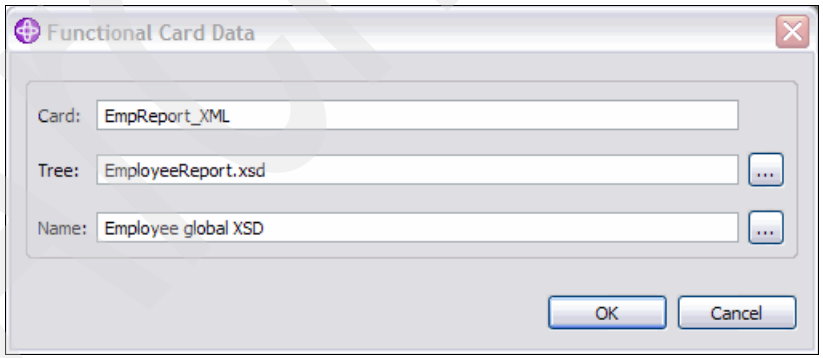

Figure 5-101 Edting a card in the Functional Map Wizard

You can also set or alter the type tree used for the card and the type within the tree (the Name field in the Functional Card Data window, instead of Type as used on a regular card). Click **OK** to save your choices.

After a map is successfully created, an icon is displayed (a green smiley face) next to the map name and the cards for that map.

6. After the cards are properly defined, click **Create** to create the functional map (Figure 5-102).

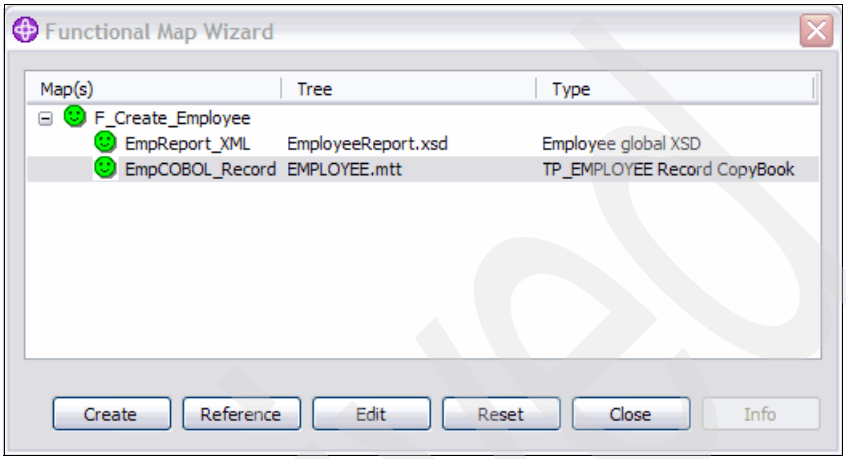

Figure 5-102 Creating a functional map

The map is created in the active map source file. You can then modify the functional map and enter the appropriate map rules.

The functional map is removed from the list in the wizard (Figure 5-103).

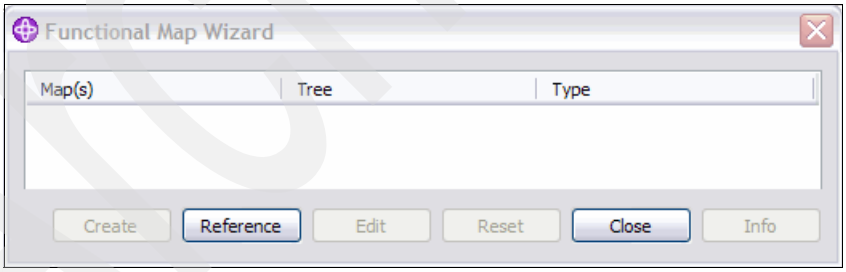

Figure 5-103 Functional map created

Continue these steps until you create all the maps that you want to create with the wizard.

7. Click **Close** to close the wizard.

Tip: Name the input cards and output cards for functional maps with the same considerations as for any input or output card, so that the card name represents the data.

Figure 5-104 shows the possible icons for cards in the Functional Map Wizard.

Meaning	Action to take
Card is undefined. The card name, type tree file, or type name is invalid.	In the Functional Map Wizard, specify a correct or valid card name, type tree file, or type name
Invalid input argument in a map rule.	Exit from the Functional Map Wizard and change an invalid input argument in rule.
Everything has been properly specified for given card or map	No action necessary. It is valid.

Figure 5-104 Icons for cards in the Functional Map Wizard

Notice that the composition of maps is displayed in the Composition view. A map that is used as a functional map is displayed underneath the map that calls it (Figure 5-105).

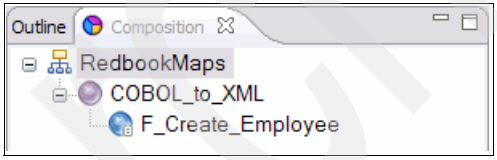

Figure 5-105 Map Composition view

The Outline view shows all maps and their input and output cards, which represent functional maps with a different icon from executable maps (see Figure 5-106).

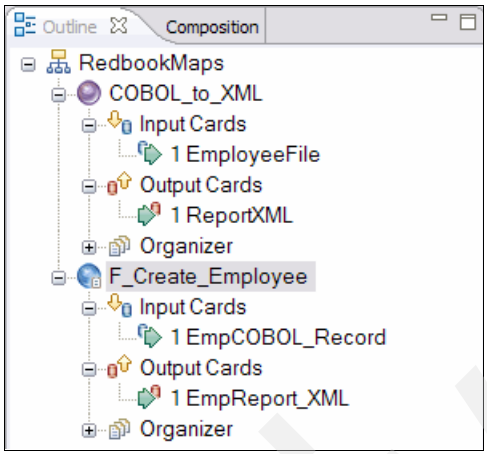

Figure 5-106 Map Outline view

To open a functional map (or any map within the map source), double-click the map name in either the Outline view or the Composition view. After creating the functional map, you have an empty functional map that is ready for you to map its rules (Figure 5-107).

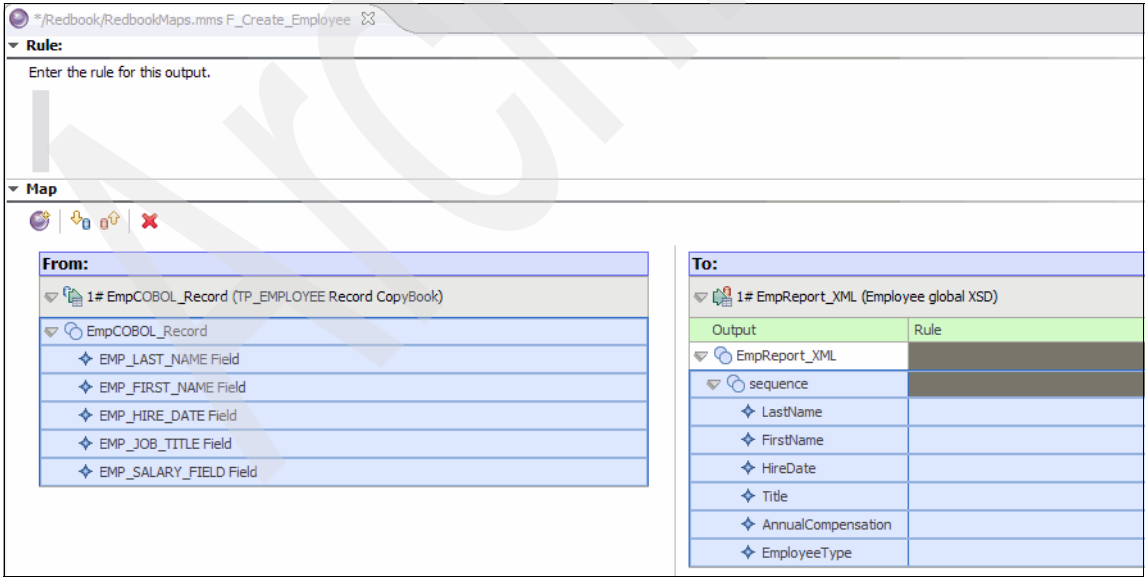

Figure 5-107 Functional Map Editor view

Create map rules for the functional map just as you do for an executable map. After you have completed all map rules, you are ready to build and test the map. Maps are not executed directly from a map source file. You must first build (compile) a map before you can execute it.

Map source file differences

Map source files contain maps, and maps contain cards. Map source file differences are compared for each map source file at the map and card levels.

Map source files are considered “different” in the following situations:

- A map exists in one map source file and does not exist in the other map source file.
- Any of the maps are different.

Maps are considered different if any of the following differences exist:

- Any card is different.
- Any card exists in the first map and does not exist in the second map.
- Any audit setting is different.
- Any remark is different.
- Any unresolved rule is different.
- The order of the cards in the map is different.

Cards are considered different in the following situations:

- The card settings are different.
- The map rules on output cards are different.

In addition to the map source file differences, the source file differences feature identifies differences for type trees, system definition files, and database/query files. You can view the results of the map source file differences.

To compare map source files in the Map Editor:

1. From the Design Studio menu, select **File** → **Map Source Differences**.
2. In the Select First File window, browse your file system and select the first map source file to be compared. Then click **Open**.
3. In the Select Second File window, browse your file system and select the second map source file to be compared. Then click **Open**.

The progress of the comparison is shown briefly in the Progress Information window, and then the Map Source File Differences window is displayed (Figure 5-108).

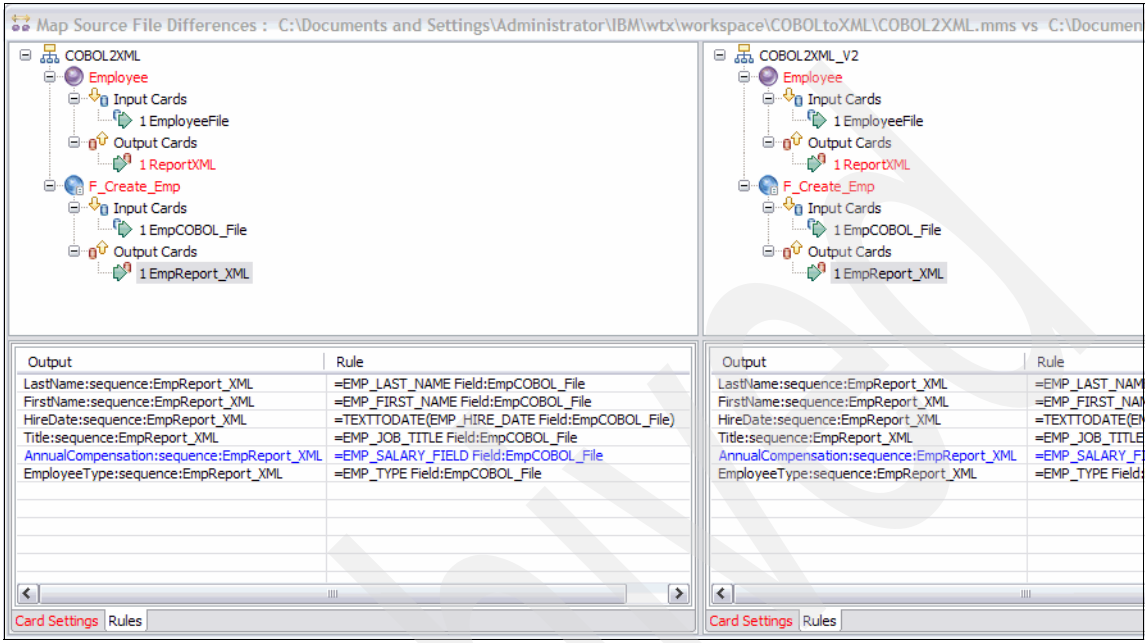

Figure 5-108 Map Source Differences window

You can resize and position the window as desired to view all differences. Maps and cards within maps with different names can be compared.

For the map source files themselves, you can compare the map settings, any unresolved rules, any remarks stored, and data audit settings. For each map, you can compare card settings for both input and output cards, and you can compare the rules for the output cards.

Colors are used to indicate the status of the differences, with differences shown in red and additions shown in blue.

5.3.5 Map settings

Map settings specify properties for map executions that are not specific to a particular input or output. These include settings for auditing a map, tracing data content, performance settings, validation settings, and map-specific warnings and errors.

Map settings are defined for a map during the design phase and are compiled with the map. The map settings that are compiled with that map can be overridden at run time (see Figure 5-109).

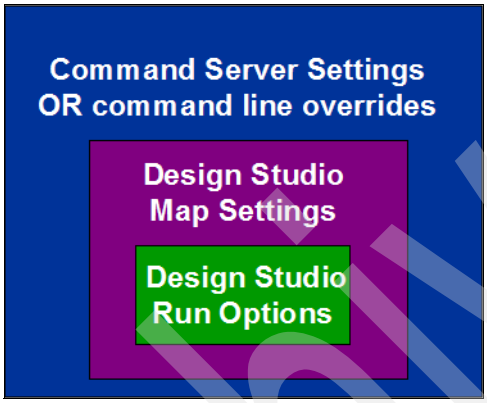

Figure 5-109 Overriding map settings

To access the map settings (Figure 5-110) for a map, from the Design Studio menu, select **Map** → **Map Settings**.

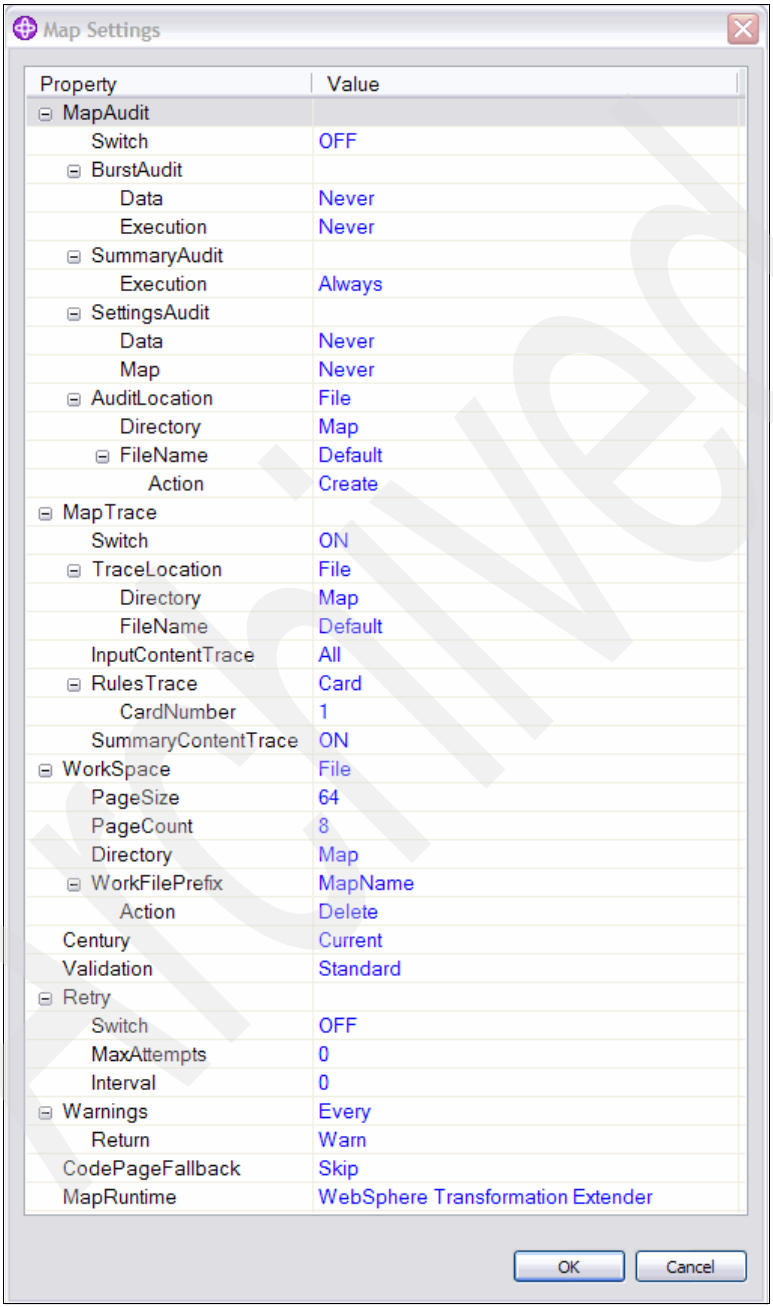

Figure 5-110 Map Settings window

MapAudit settings

MapAudit settings specify options for creating and storing map audit logs. If the MapAudit Switch is set to ON, the audit log is created and can be appended. In addition, with flexible subswitches, the audit log can be created or appended based on the following conditions:

- A map fails.
- The map has an error.
- The map has a warning or error message.

You control whether audit data is generated, what information is included in the audit data, and where the audit data is stored.

MapTrace settings

MapTrace settings specify properties for a trace file that are used to diagnose invalid data or incorrect type definitions. You control whether a map trace is generated, what information is included in the trace, and where the trace data is stored.

WorkSpace settings

WorkSpace settings specify the location (file or memory) of the workspace and PageSize and PageCount settings for configuring the page size. By using WorkSpace settings, you can optimize the speed of map processing.

When a map is run, a paging scheme to access the data as efficiently as possible is used. Portions of both working data and actual data are paged as needed. The workspace can be a combination of files and memory, or simply memory-based. When a file-based workspace is used, the amount of memory accessed at a given time depends on the number of pages and the count of pages configured for a map to run.

Official advice for page size is not to change the default (8 pages of 64 KB) unless the map run returns a paging error (#18-Page usage count error, #22-Page size too small, #34-Too few pages requested, more allocated, and #35-Insufficient number of pages to execute map).

Our experience shows that the workspace and work area can change map performance dramatically. Therefore, adjust the workspace and work area settings whenever you experience performance issues or you must optimize a map. First you must determine whether you should work on memory or file workspace.

Memory workspace avoids disk I/O access time by allocating random access memory (RAM) for the transformation process. This implies that the memory allocation need is less than the physical available memory. Otherwise, the

operating system memory management simulates additional memory with disk storage and this ultimately degrades performance. You might consider using memory workspace whenever you are dealing with small amounts of input data.

Large amounts of input data are handled better with a file-based workspace. Then it is essential to properly size the work area to the average input data size. First, determine the average size of input data that the map is going to process. Then adjust the size of the page to the amount of objects per record. The more components there are (that is, the more data objects there are per occurrence of record), the bigger the page is required.

Mapping also has a great influence on the need for page size. Data structures influence the size of each temporary file for each input and output card. If the record group is fixed size, the temporary workfile for the related card is more or less the same size as the data itself. Other group formats require three to five times the size of the data, depending on the properties of the group and, for an output card, the type of mapping.

Next, adapt the amount of allocated pages. The more records you have, the more pages you should allocate. To test page settings, you can use the **-p** execution command option so that you can specify the page size and page count for a specific run. By using the **-w** options, you can specify the workspace settings. With this option, you can specify whether you want to use memory (M) or file (D or DU for unique file) workspace (Figure 5-111).

The example in Figure 5-111 allocates 16 pages of 128 KB and uses a file workspace for a run of the swift2xml.mmc map.

```
dtxcmdsv.exe swift2xml.mmc -p128:16 -wd
```

Figure 5-111 Command line to run swift2xml.mmc and override workspace settings

Check and compare the time spent for each run (see ElapsedSec of the execution summary segment in the audit log) to determine which settings allow the quickest run. Remember that an optimal page setting for a given map and data file might not be optimal for the same map with a much larger or smaller data file.

WTXPAGE

The WTXPAGE tool is included in an installation of the run time that generates an iterative search of the best workspace settings. You enter the **wtxpage** command followed by the path and the name of the map that you want to optimize. The command server runs the map several times with specific workspace settings each time. WTXPAGE returns the settings that generated the quickest run.

Figure 5-112 shows a sample report from a WTXPAGE execution. Before using WTXPAGE, run the map several times to ensure that your environment is stable. The time spent to run the map should remain equivalent from one run to another. If not, stabilize your platform before running the tool.

```
C:\IBM\WTX8.2>dtxpage C:\$Redbook\MT103MsgSWIFTToXML.mmc

IBM WebSphere Transformation Extender
Page Setting Assistant for Maps - version 8.2(58)
¬ Copyright 2008 IBM Corporation. All rights reserved.

.size  128      count   16      time 3128590
.size   64      count   16      time 3135235
.size  128      count   16      time 2986540
.size  256      count   16      time 2943324
.size  512      count   16      time 2900243
.size 1024      count   16      time 2908623
.size  512      count   16      time 2894945
.size  512      count   16      time 2859252
.size  512      count   12      time 2920007
.size  512      count   16      time 2869286
.size  512      count   20      time 2928737
.size  512      count   16      time 2864007
.size  512      count   12      time 2913045
.size  512      count   16      time 2871640
.size  512      count   20      time 2914473


Optimal page settings for C:\$Redbook\MT103MsgSWIFTToXML.mmc are
SIZE: 512  COUNT: 16


C:\IBM\WTX8.2>
```

Figure 5-112 Using WTXPAGE to determine the best page settings for SWIFTToXML.mmc

Important: You *must* run WTXPAGE from the path where it is located. That is the home folder of the WebSphere Transformation Extender installation for Windows and the bin folder for a UNIX or Linux installation. WTXPAGE also requires that the map runs successfully (return code 0) and that no execution command options are used.

WTXPAGE is only a tool to help your decision, and its result must be properly interpreted. For example, if a CPU is heavily used during a specific iteration, it alters the iteration result. Run WTXPAGE several times and see what results show more frequently. Also, the quicker the map runs, the less reliable the result is.

Century setting

The Century setting specifies the current or sliding century.

Validation setting

The Validation setting specifies how strictly the data is validated by specifying a response behavior for types of validation errors.

Retry settings

The Retry map settings define whether to try again if compiled map files, work files, audit log files, or trace files that are needed during map execution cannot be opened.

Specify the Interval and MaxAttempts to define the number of times and at what interval attempts are made to access an unavailable file.

Warning settings

Warnings settings specify that the map should fail when any of the selected warning codes are returned. The default value for the Warnings setting is *Every*.

Enabling the Warnings options affects the transaction behavior specified for each card in the map. If a map execution warning code occurs, the map fails after processing of that map is finished.

Maps called (from the failed map) with a RUN function, PUT function, or GET function still process. The behavior defined in that map's card settings for Transaction OnSuccess and Transaction OnFailure occurs.

If any warning is set to fail the map, and the map fails due to the cause associated with the selected warning, the execution audit includes the map-failed message with the appropriate warning code preceded by a minus sign, for example, `Map failed on warning (-27)`.

CodePageFallback setting

The CodePageFallback map setting is used to specify, on each map, the code page conversion fallback behavior when mapping between objects with specific data languages set and an unknown character is found during the conversion process. The CodePageFallback setting has the following options:

- SUBSTITUTE for using the International Components for Unicode (ICU) substitution character specified in the conversion tables
- SKIP for skipping characters that cannot be converted to the target code page

MapRunTime setting

The MapRunTime setting determines the runtime platform of an executable map. The default value is WebSphere Transformation Extender, which is the standard method. The alternate option, WebSphere DataPower, pertains to a WebSphere DataPower SOA Appliance.

5.3.6 Compiling maps

After you enter map rules in an executable map, create necessary functional maps, and enter map rules for the desired output data objects, you must build the executable map. A compiled map file is created when no build errors occur while the executable map and all referenced maps are analyzed.

An executable map has a data source or target specified for each of its cards. Executable maps are built. Remember that functional maps are not built.

Data sources and targets specified for an executable map are defaults built into the compiled map file. However, data sources and targets, and other map settings, can be overridden at run time.

When a map is built, the map and all of the functional maps that are referenced within that map are analyzed. For example, when the map called MyMap, which references MakeRecord, which references ItemMap, is built, all three maps are analyzed. If there are no build errors, a compiled map file is created. The referenced maps are compiled within this file. Referenced maps are not created as separate compiled maps.

Use the **`Build`** command to compile an executable map. The **`Build`** command analyzes the logical interfaces within map rules, and if the analysis is successful, it compiles the map. Analysis includes checking for missing rules, invalid rules, invalid card definitions, verifying map references, and verifying the arguments of functions. Analysis also looks for *circular map references*, maps that reference one another. You can view any resulting build errors or warnings on the Build Results tab of the Organizer.

The following build options are available:

- Build

  Use Build to build the selected executable map for a Windows operating system. When a DataPower map is selected (by using the map settings), the map is built for the DataPower appliance.

- Build all

  Use Build All to build all maps within the map source file for a Windows operating system. When a DataPower map is encountered (by using the map settings), that map is built for the DataPower appliance.

- Build for Specific Platform

  Use Build for Specific Platform to build the selected executable map for an operating system (platform) other than Windows.

- Build All for Specific Platform

  Use Build All for Specific Platform to build all maps within the map source file for an operating system (platform) other than Windows.

To build a map for Windows, open the map to be built, and from the Design Studio menu, select **Map** → **Build**. The Building Map window is displayed briefly and closes automatically.

Build results are displayed on the Build Results tab on the editor. An empty Build Results tab indicates that there were no errors and the map compiled successfully. Error or warning messages that occur during the map build process are displayed on the Build Results tab. If a build error occurs, a map cannot be compiled until the problem is corrected. That is to view the errors, correct the problems, and build the map again. A map cannot be run if it has not been built.

Any unresolved map rules are displayed on the Unresolved Rules tab on the editor. An unresolved map rule happens when the type of output card is changed and there was a map rule on an object that no longer exists in the card object. As a result, that map rule becomes unresolved.

Unresolved map rules are retained for reuse or modification. Unresolved map rules can be dragged to a new output or deleted. There might be times when you enter a long or complicated map rule that becomes unresolved. Use the drag-and-drop capability to save time and reduce the risk of errors.

Maps that are compiled for specific platforms are compiled with platform-specific file name extensions. The name of the platform-specific compiled map file is the executable map name with the platform-specific extension, such as the .aix extension. For example, building the map “MyMap” for the Linux platform creates

the `MyMap.lnx` file in the map directory. Refer to Figure 5-19 on page 187 to see where you set the default map extensions.

These default file name extensions prevent you from inadvertently overwriting your original compiled map and help identify the compiled map file to transfer to the specific platform environment.

Gathering compilation information: No reverse compiling tool is currently available for WebSphere Transformation Extender, making it difficult to get information about what a compiled map contains. It is crucial on a project to keep a mirror copy of every compiled map that is deployed on each environment and the related sources (preferably in a versioning tool).

However, we can use the following tricks to obtain more information about compilation:

- Keeping the extension (or any other kind of reminder in the map name) indicates the platform for which the map has been compiled.
- Indicate in which version we have compiled the map. See “Inserting a version in the map” on page 668.
- Reading the compiled map with a HEX editor provides information. For example, Byte#200 (8 - cOh) is the trace and log setup. The first four bits are the trace (A=off, E=on) and the last four bits are the log (0=off, 1= on) settings. Therefore, if Byte#200 is A0, then the map has no trace nor log activated.

Remember that the ascending compatibility between run time and a compiled map. A map compiled in a V-old version will run properly with a V-new run time (Command Server and Launcher).

Compiling from a command line

The **`mcompile`** utility command permits the compiling one or more maps from a command line, outside the Design Studio GUI. The **`mcompile`** utility command returns 0 if the compilation is successful and 1 if it is not successful. It is a batch operation that can also be used for automation.

A map source file (.mms file) is a required field. It is the name of the map source file that needs to be compiled. The map source file can have any number of executable maps in it. The executable maps must be defined as a map for which all sources and targets have the minimum amount of information specified for the actual source or target of data for that card’s data. If the full path of the map source file name is not specified, the **`mcompile`** utility command searches for the map source file in the current directory.

In the **mcompile** utility command, all the options, except for the executable map name, are case insensitive. Example 5-5 shows the syntax of the **mcompile** utility command.

Example 5-5 The mcompile utility command

```
mcompile <.mms file> -L [-DP | -TX]
   - or -
mcompile <.mms file> -A
   [-K] [-E] [-NO]
   [-DP] [-TX]
   [-P <platform>]
   [-O <.mmc file name/location>]
   [-R <.mme file name/location>]
   [( -LOG [.
     | <log file name/location>]
     [-FAIL] [-VERBOSE] [-APPEND])
   | -NOLOG]
   - or -
mcompile <.mms file>
     <executable map>
   [-E]
   [-NO]
   [-P <platform>]
   [-O <.mmc file name/location>]
   [-R <.mme file name/location>]
   [( -LOG [.
     | <log file name/location>]
     [-FAIL] [-VERBOSE] [-APPEND])
   | -NOLOG]
```

See the WebSphere Transformation Extender documentation for details and examples of the **mcompile** command.

5.3.7 Integration Flow Designer

The Integration Flow Designer is a component of the Design Studio that provides a graphical facility to combine collections of maps and run them as a single unit of transformation. The Integration Flow Designer has its own easy-to-use graphical interface. You can create system diagrams and what-if scenarios, generate process control information, and see how the systems will behave at execution time.

Additionally, the automation of the Integration Flow Designer helps to reduce the number of errors that often result from manual operations. It shows, in plain graphics, what your system is doing (useful for documentation and presentation purposes), prepares your system for runtime environments, and deploys it to various platforms because it offers an FTP deployment facility. It also validates system logic consistency and generates the command server command file or launcher control file that will be used in the runtime environment.

Integration Flow Designer basics

Because Integration Flow Designer is not currently Eclipse based, you can start it from the Design Studio menu. From the Windows desktop, select **Start** → **All Programs** → **IBM WebSphere Transformation Extender 8.2** → **Design Studio** → **Integration Flow Designer** (Figure 5-113).

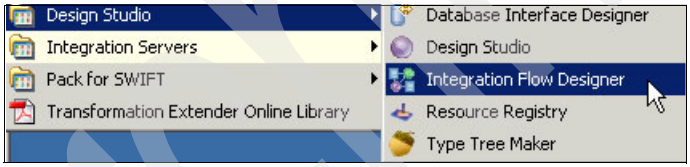

Figure 5-113 Starting the Integration Flow Designer from the Design Studio Start menu

The Integration Flow Designer has distinguishable areas as highlighted in Figure 5-114.

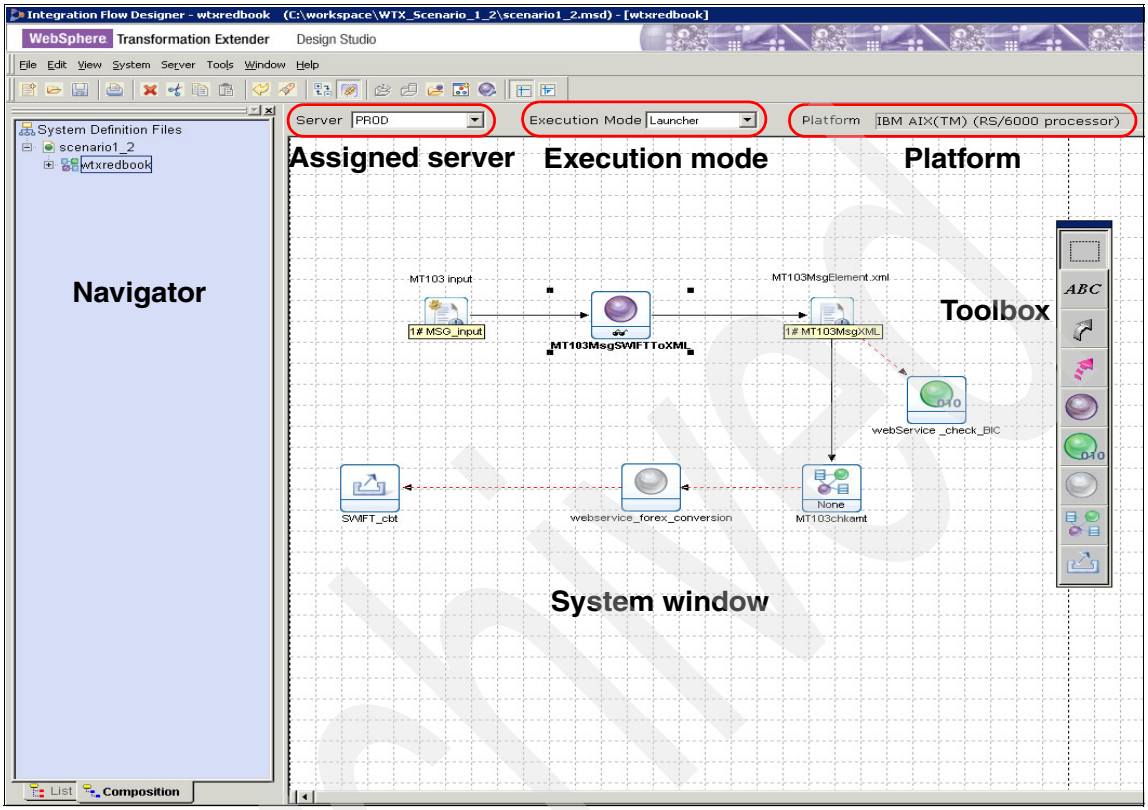

Figure 5-114 Integration Flow Designer navigator, system window, and toolbox

We explain each of these areas of the Integration Flow Designer as follows:

- The *navigator* (Figure 5-115) shows the open system definition files (.msd) and their contents, in either the List or Composition view.

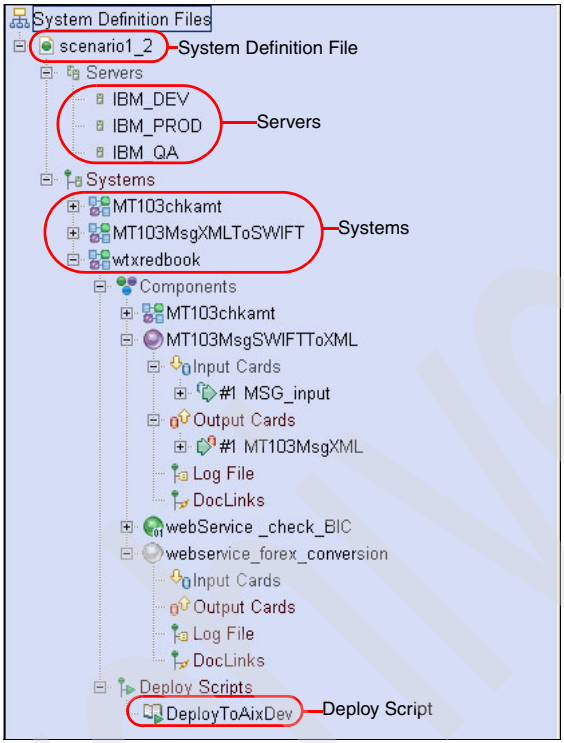

Figure 5-115 Navigator List view of the Integration Flow Designer

- The *system window* shows a graphical representation of the components of the selected system.

- The *toolbox* provides tools (Figure 5-116) that are needed to create or modify system diagrams.

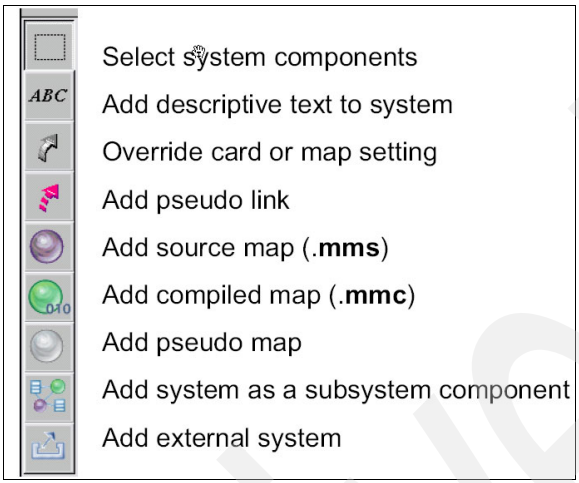

Figure 5-116 Toolbox icons and their properties

- *Assigned server* is the server on which you expect the system to run.
- *Execution mode* is the runtime method that you are willing to use for the system.
- *Platform* identifies the assigned server hardware and operating system.

Preparing the template system definition file

You usually start by creating a template system definition file (.msd) that contains server definitions. This templates contains server information that can then be used for any new system definition. Such information includes hardware, operating system, FTP user ID and password, and host name or IP address.

To prepare the template system definition file:

1. Create and save a new system definition file. Select **File** → **New** and then press Ctrl+S to save this system definition file with any logical name (for example `system_template.msd`).

2. Add all the servers one by one:
   a. Select **Server** → **New**.
   b. In the Add Server window (Figure 5-117), specify the following information and click **OK**:
      - Server logical name
      - Server platform
      - Server FTP user ID and password
      - Server IP address or host name

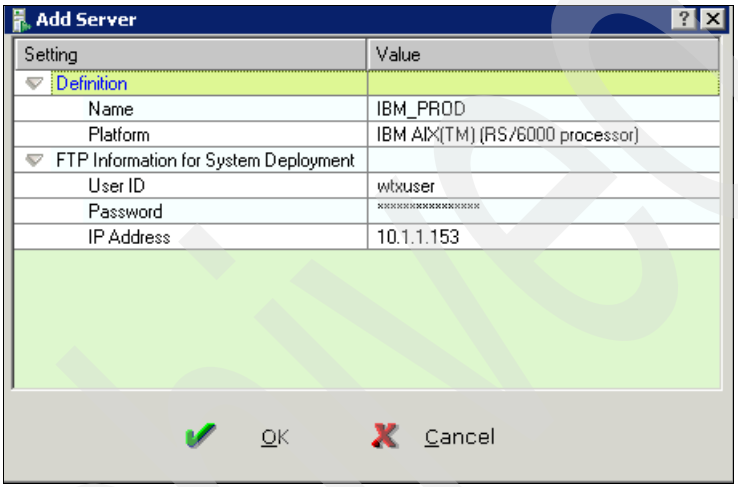

Figure 5-117 Add Server window

The new server is added to the server list.

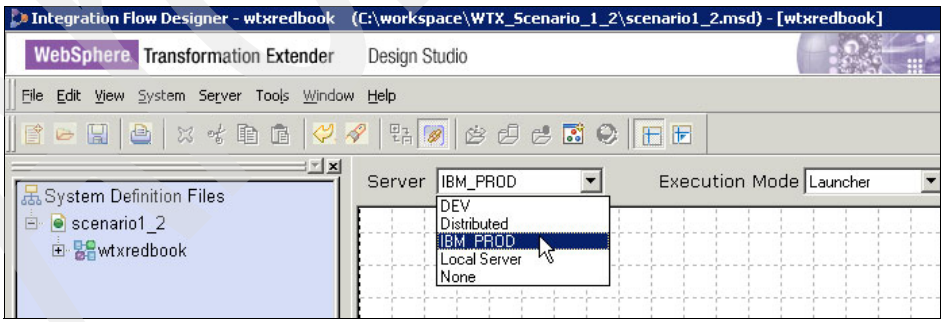

Figure 5-118 New server IBM_PROD in the Server list

You can do the same for all servers that are used in your environment and then save this file as a template. This way, each time you have a new system definition file to create, it already contains all information about your servers.

Note: It is likely that, while you have direct FTP access to development servers, you unlikely have it on QA nor on preproduction and production. However, it is possible that FTP access is granted on a “parking area” of those servers (either a DMZ or FTP dedicated server), where you still have to do system delivery.

Setting Server and Execution Mode properties of your system

Before anything else, set the execution mode (Launcher or Command Server) and the assigned server of your system (Figure 5-119). This implies the platform because it is a property of each server.

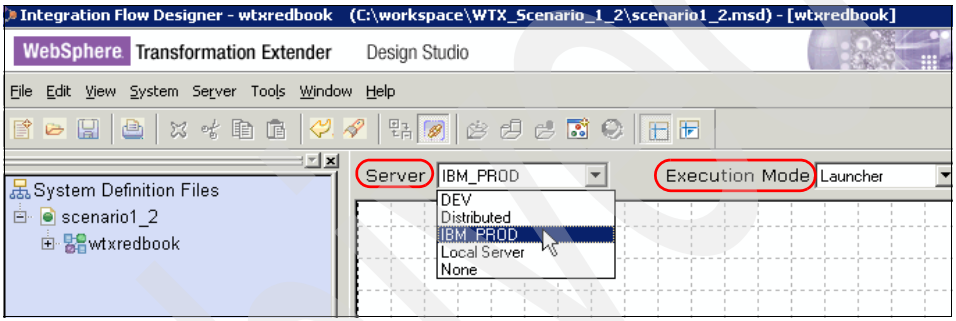

Figure 5-119 Setting the destination server and Execution Mode

Adding components to your system definition file

To add a component to your system, use the toolbox and add the relevant object. You *must* use the toolbox because it is the only way to add any component.

Displaying the toolbox: If the toolbox is not displayed in the System Window, select **View** → **Toolbox**.

You can add three types of maps to a system:

- A *source map* references an executable map that is defined in a WebSphere Transformation Extender map source file (.mms). It is possible to override most of the properties of such maps, and Integration Flow Designer links to Design Studio if you need to modify the mapping.
- A *compiled map* references maps that are already compiled. Although it sounds easier to manage such maps, there is one negative side. Since no reverse compilation is possible, it is impossible to modify the mapping of such component. Instead, deal with source maps.

- A *pseudo map* acts as a placeholder for a map that has not yet been completely defined. You need a map in your system, but it is not defined yet. Integration Flow Designer can assist in the creation of the map when you are ready to create it.

There is no restriction to adding elements. A system can be as simple (only one map) or as sophisticated (multiples maps and subsystems) as you want. Keep in mind that they represent a unit of transformation.

A map added to a system is called a *map component*. A map component represents an executable map and shows its relationship toward other components of the same system. It has five basic characteristics:

- Name
- Class (source, compiled, or pseudo map)
- Execution settings (execution mode, trigger if any)
- Source and destination interfaces
- Document links

To add a map to a system from a map source (Figure 5-120 on page 287):

1. Open a target system window.
2. Click the **Add Source Map** tool.
3. Click the target system window.
4. In the Add Source Map window:
   a. Select the map source file name (.mms file) from the browser.
   b. Select one or more executable maps.
   c. Click **OK**.

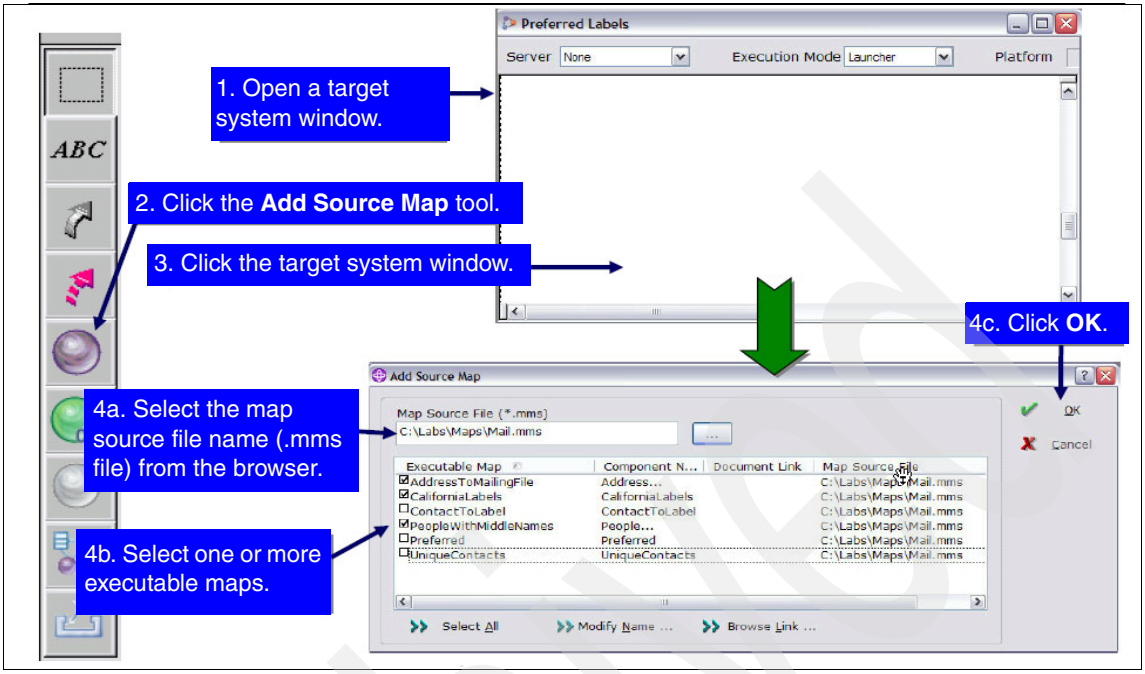

Figure 5-120 Adding references to source maps

A map component is added in the contracted view. Double-click the map icon to expand the view and see the card information. Data dependencies among map components are represented by internal links that show the data flow direction.

By using the expanded view (Figure 5-121 on page 288), you can perform the following actions:

- View card details. Right-click the card and select **View Card Details** (viewer only).
- Modify card definitions. Right-click the card and select **Open Type Designer**, which links you to the Design Studio Type Tree Editor with the type tree of the selected card.
- View the source and target definition. With the cursor, hover over the card for a few seconds to see the command line of the selected card.
- Override the source and target. Double-click the card to override all settings.
- Add document links. Right-click and select **Doc Links** to add any type of documentation.

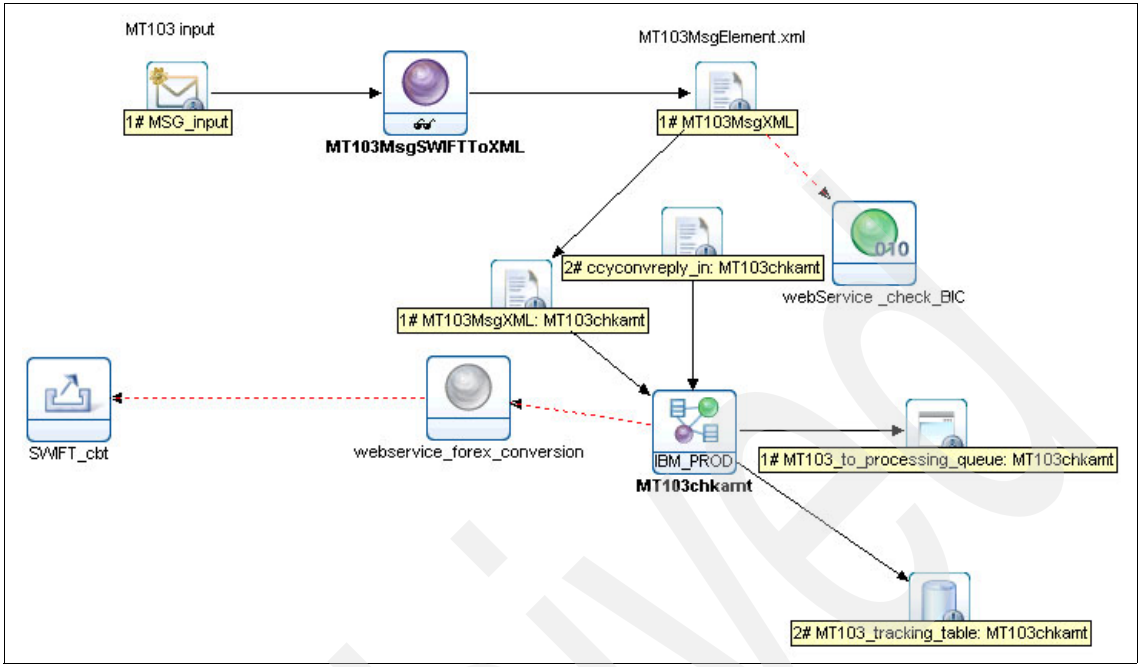

Figure 5-121 Viewing card information in map component expanded view

Changing a map name

Changing a map name can be required. It is especially useful when you want to use the same map source to add two instances of the same executable map with specific overrides for each. Right-click the map component and select **Edit Map Component**. In the Edit map Component window (Figure 5-122), you can change the Component Name. Executable Map Name has a list from which you can choose the map that you want from the selected Map Source File.

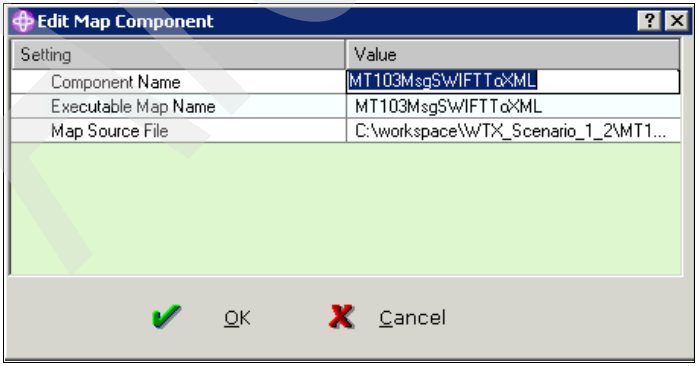

Figure 5-122 Edit Map Component window for a source map

Note: The settings that are shown in Figure 5-122 on page 288 are for a source map. For a compiled map, you can change the Component Name and the Executable Map File, while a pseudo map has only a Component Name.

Modifying mapping of a map component

To open the map in the Design Studio, right-click and choose **Open Map Designer**. Make the appropriate modifications and save the map source to see the updated map in the Integration Flow Designer.

Modifying component properties

The first property that you might modify is the MapServerLocation setting. The MapServerLocation setting identifies the path and the name of the compiled map file *on the server*. The default value for the MapServerLocation setting is the path of the selected map *on the development machine*. This path is where the Launcher expects to find the compiled map. This value affects the Server View of the Server Map Directories section of the deploy scripts definition.

Launcher execution mode

After adding components to your system, you can change their properties. The properties of a WebSphere Transformation Extender map have several levels of settings. See 3.3, “Maps” on page 72.

You can change the properties by right-clicking a component and selecting **Edit Launcher**. You can also double-click the **Execution Settings** button without altering the properties of the executable map at the Design Studio Map Editor level. See Figure 5-123 on page 290. The same transformation map can have different settings depending on where it was compiled from (Design Studio Map Editor or Integration Flow Designer). Here, we deal with a specific instance of an executable map, coming from a map source file.

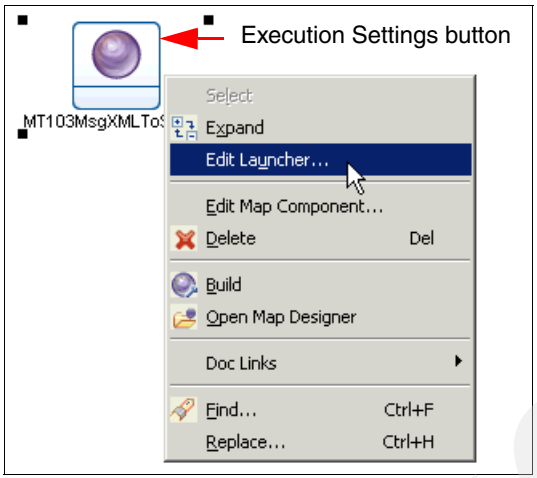

Figure 5-123 Execution Settings button and the Edit Launcher pop-up menu

In the Launcher Settings window (Figure 5-124 on page 291), you can change all properties of the selected component map. You can change all map properties, but you also can select the execution method of each component.

If you want to benefit from multithread capacity of the Launcher, you must ensure that each component that should have parallel runs is able to do so. The components must not share any resources. Otherwise, each instance is placed in pending mode until the current running instance finishes execution. That is, it will run in sequential mode. At a minimum, you must specify the following information:

- Unique WorkFilePrefix (if the default File WorkSpace is used)
- Unique log file

For Map Component, you can specify the time event or source event, and the event coordination, that will constitute the Watch of the map.

Note: On WebSphere Transformation Extender, a *trigger* is an event that contributes to the initiation of a map. The set of events that initiate a map is called a *watch*. A watch is the same as a sum of triggers that are defined for a map initiation.

The source event can be either or a combination of the following types:

- A change or appearance of a file
- A message in a queue
- An insert or update of a database (Oracle and SQL Server only)

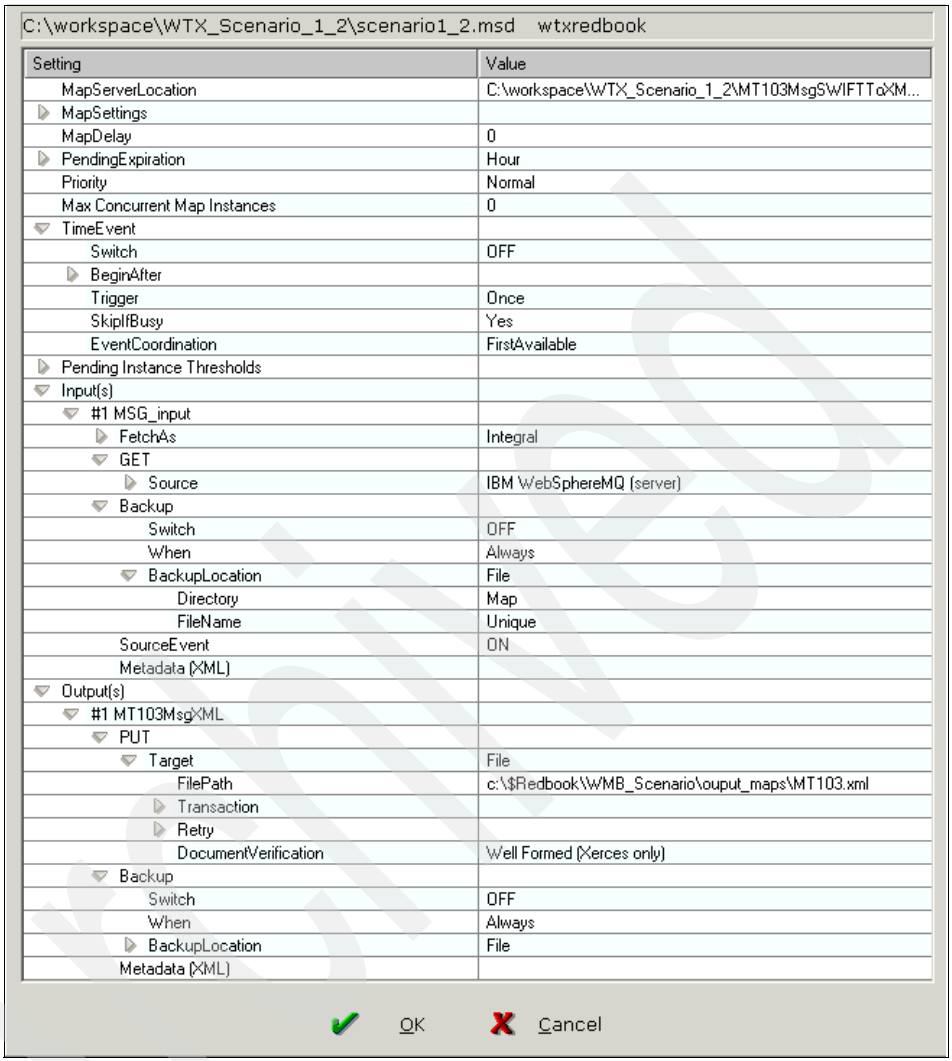

Figure 5-124 Launcher Settings window

Command execution mode

You can also edit component properties in command execution mode, but only the map settings.

Preparing system for run time: Launcher mode

You must specify a watch for each map, except for those that will be called by a RUN function from another executable map. You can set the time event, source event, or both.

Source Event

In the Launcher Settings window (Figure 5-125), expand the **Input(s)** properties.

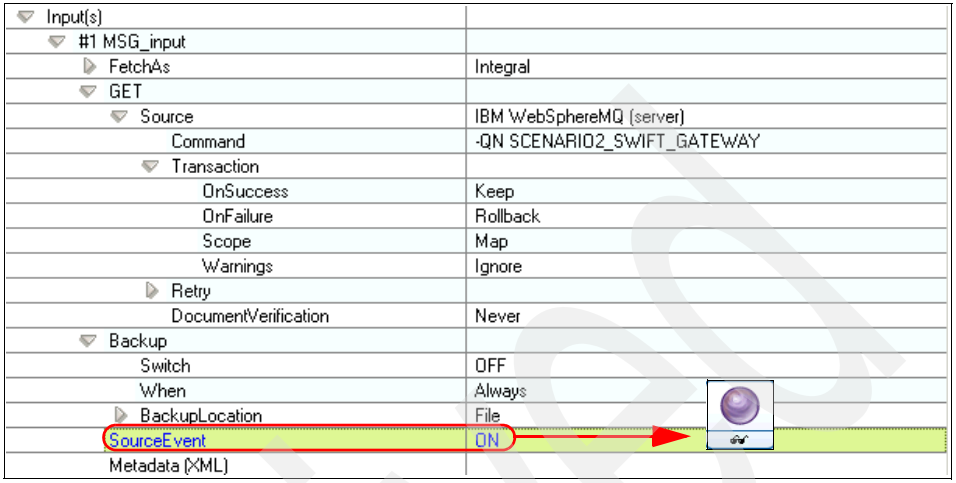

Figure 5-125 Input(s) properties from the Launcher Settings window

Each input card is represented. Expand the appropriate input card to switch SourceEvent property to ON. A source trigger icon (sunglasses) is displayed in the Execution Settings button of the component map.

You can set up several input cards as source events. If you do, they must all be triggered for the map to be initiated.

You can also use wildcards in the SourceEvent setting. The asterisk (*) replaces any set of characters, while the question mark (?) stands for one single character.

You can also use wildcards for output cards. You can set up wildcards only if they are also used on an input card so that they then have the same value as the input card. Therefore, if you use wildcards on several input cards of the same map, they must have the same value for the map to be initiated.

For example, you can set up SourceEvent on input card #1 with a *.txt value and set up output card #1 with an *.xml value. If the customers.txt file is displayed, the map is initiated and output card #1 generates the customers.xml file. Also, if input card #2 has a *.cfg value, the map waits to have a customers.cfg file available before it is initiated.

Time Event

In the Launcher Settings window (Figure 5-126), expand the **TimeEvent** properties.

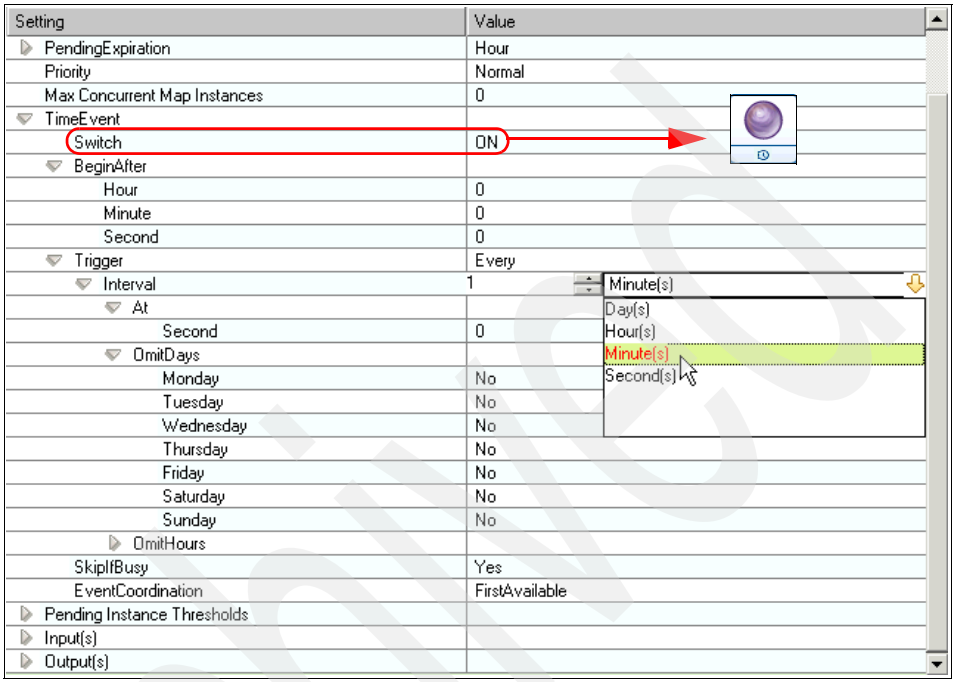

Figure 5-126 TimeEvent properties from the Launcher Settings window

The purpose of the TimeEvent properties is to schedule the component map initiation by using the Launcher scheduler. BeginAfter is used to execute a map a certain period after system initiation. With Trigger settings, you can specify a proper time schedule with interval, interval unit, and omits.

You can specify a Source Event combined with a Time Event (Figure 5-127). The EventCoordination determines whether the map will be initiated if one of the events or all of the events occur. EventCoordination only affects behavior between a Time Event and Source Event.

Figure 5-127 Button showing Symbol for Time Event combined with Source Event

Preparing system for run time: Both modes

You can now prepare a deployment script to transfer the system control file and its components:

1. Select **System** → **Deploy** → **Definitions**.
2. Add a logical name.
3. Transfer all component maps to the Server View directory. (The default setting is to build all component maps.) Then generate and transfer the Launcher Control File (or Command Control File in Command Server Execution Mode) to the assigned server in the same path as it is in the System Definition File. Most of the time, the deployment script should be modified:
   a. Double-click **Build and Transfer Maps**.
   b. Click **Map to Process** to specify whether you want to process all maps or only some of the maps.
   c. Click **Server Map Directories**. Server View shows what you defined in “Modifying component properties” on page 289.
      i. Specify the Integration Flow Designer View, which is the deployment path of our components from the root of your FTP access.
      ii. Right-click and select **Browse Server Directories** to browse the target location.
      iii. Click **OK**.
      iv. Click **OK** to close the Server Map Directories settings.
   d. Double-click **Generate and Transfer Launcher Control File** (or **Generate and Transfer Command File** in Command server mode).
   e. Click the **...** button to specify the path name of the Launcher Control File (or Command File) on the destination server. Click **OK**.

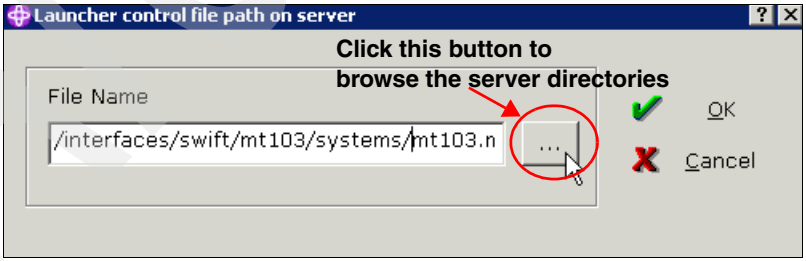

Figure 5-128 Launcher control file window

f. In the Define Deploy Scripts window (Figure 5-129), select **Transfer Additional Files** if needed. Any file can be added to the deployment script, including resource registry naming files (.mrn files), XSD files, data files, and a `dtx.ini` specific file. Specify the transfer mode as either **ASCII** or **Binary**.

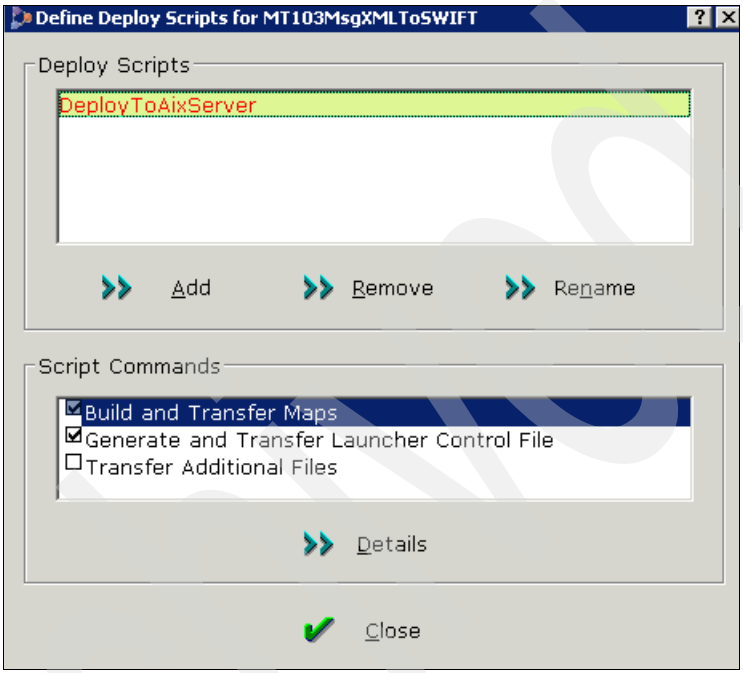

Figure 5-129 Define Deploy Scripts window

Deploy script: A deploy script is not mandatory. In quality assurance and production environments, it is unlikely that system administrators grant you FTP access. You can then prepare your system to be run by compiling the maps (**System** → **Build maps**), generating a control file (**System** → **Generate**), and providing those files to the administrators.

Now that each component is prepared for run time, analyze the system to see if it is ready to be deployed. Select **System** → **Analyze** and make sure that there is no error or warning message.

5.3.8 Database Interface Designer

The Database Interface Designer is used to import metadata about queries, tables, and stored procedures for data stored in relational databases. It is also used to identify characteristics of those objects to meet mapping and execution requirements such as update keys and database triggers. Any database object that is used by WebSphere Transformation Extender must be implemented in the Database Interface Designer.

The Database Interface Designer is used to perform the following tasks:

- Specify the databases to use for a source or target
- Define query statements
- Automatically generate type trees for queries or tables

After defining database queries or tables in the Database Interface Designer, you can define a map in the Map Editor, where, in the map cards, you can specify an input source as either a query or a stored procedure. In addition, you can specify an output target as either a table or a stored procedure.

Database connectivity is supported under the control of the relational database management system (RDBMS). The adapters provide the option of using a driver to connect to the platform of your choice so that you can automatically create type trees for database queries and tables. The adapters also provide a test environment on the development system for maps by using data stored in a database. You can also install adapters on additional systems to provide remote database connectivity, such as with ODBC, Oracle, Sybase, or DB2, on operating systems such as HP-UX, Sun Solaris, AIX, and so on.

A database or query file is stored as a .mdq file, which contains the definitions for one or more databases, as well as queries, stored procedures, and other specifications that can contribute to the execution of a map. We provide a high-level overview of the steps to create and use a database or query file.

To use the Database Interface Designer to import database definitions:

1. Create a database or query file (.mdq file).
2. Define a database.
3. If the database is to be used as an input, define a query for that database.
4. Generate the type tree for the query, table, view, stored procedure, or message from which or to which you want to map.
5. If the database is to be used as output, set keys and designate columns to perform SQL updates.
6. Save the .mdq file.

To use the Map Editor to configure database sources, targets, or operands in a rule:

1. From the input or output cards in the executable map, select **Database** as the value for either the GET Source or PUT Target setting.
2. Select the .mdq file that contains the database-specific source or target information.
3. Perform either of the following actions for an input or output card:
   - In an input card, select a query from the .mdq file.
   - In an output card, specify a table name or stored procedure.

Database Interface Designer basics

During installation of the Design Studio, an entry for the Database Interface Designer is added to the WebSphere Transformation Extender program folder (listed under the Design Studio).

Because the Database Interface Designer is not currently Eclipse based, you can start it through Design Studio by selecting **Start → All Programs → IBM WebSphere Transformation Extender 8.2 → Design Studio → Database Interface Designer**.

When the Database Interface Designer runs, in the Startup window (Figure 5-130 on page 298), you can select how the Database Interface Designer program should open. The following options are available:

- Open an existing database/query file. Browse for the .mdq file that you want.
- Create a new database/query file. Create a new .mdq file.
- Open a recently used database/query file. Select one or more files from the displayed file list.

You can also double-click a file from the list to open it. You can disable the Startup window by selecting the **Do not show this at startup** check box. However, you can always access this window from the Database Interface Designer window Help menu by selecting the **Startup Window** menu option.

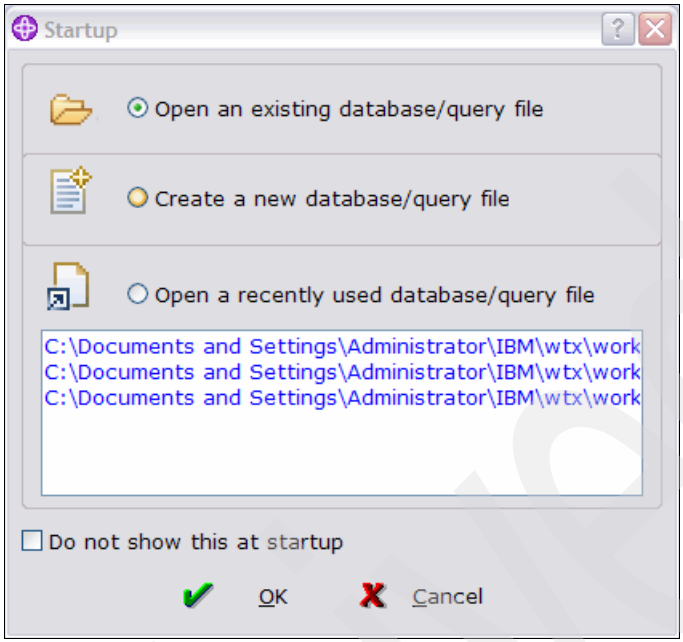

Figure 5-130 Database Interface Designer startup window

The main Database Interface Designer window (Figure 5-131) provides a graphical user interface in which to create and maintain .mdq files. When you create a database/query file in the Database Interface Designer, a file named Database_Query File (the default file name) followed by an assigned, sequential number is displayed in the Navigator, indicating that you can begin defining databases, queries, and so on.

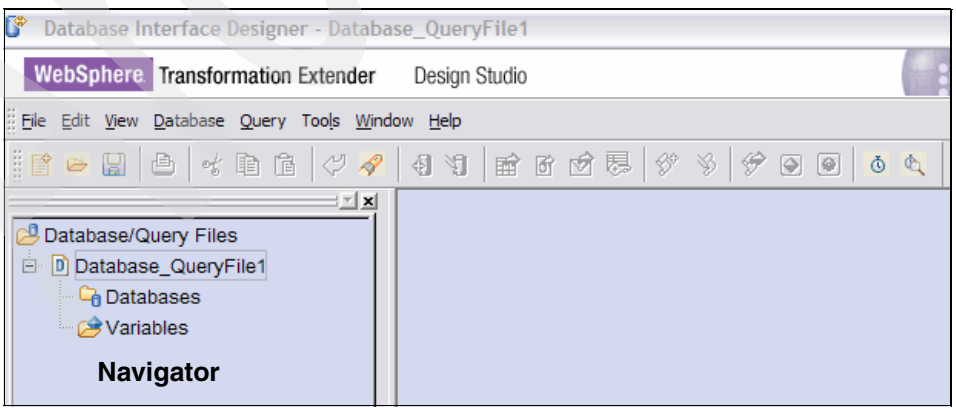

Figure 5-131 Database Interface Designer main window

Save this database or query file (a file name with a .mdq extension) by selecting **File** → **Save As** and providing an appropriate name and location.

The Navigator, in the left panel, graphically presents all of your opened .mdq files and the databases that they contain. It also provides a graphical representation of the queries, stored procedures, message queues, and tables or views that have type trees generated. Also displayed are tables and views with update keys defined, as well as variables that are defined in each .mdq file.

The Navigator can be shown or hidden. It also can be presented as either a docked window or a floating window. These choices are available from a pop-up menu in Navigator. To access the pop-up menu, right-click the top border of the Navigator (Figure 5-132) and make your selection.

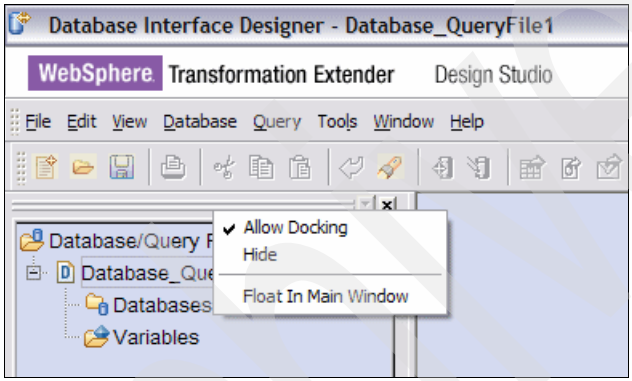

Figure 5-132 Database Interface Designer Navigator window options

When a .mdq file is displayed in the Navigator, you can add new database definitions to it or modify the name of an existing database, which you do in the Database Definition window (Figure 5-133). Each Database Definition window contains some settings that are common across all platforms and other settings that are platform-specific.

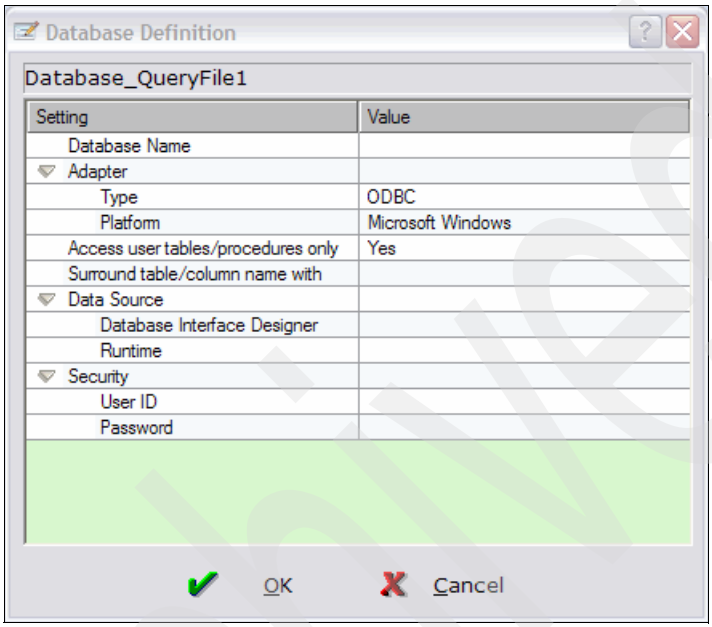

Figure 5-133 Database Definition window

To define a database (Figure 5-134):

1. From the Navigator, under the .mdq file, select **Databases**.
2. From the Database menu, select **New**, or right-click the Databases object in the Navigator and select **New**.
3. In the Database Definition window, enter the information for the remaining settings as desired. See the explanation following Figure 5-134. Then click **OK**.

More information: To view more information about each platform-specific setting, see either the context-sensitive help that is available from the window itself or each platform-specific reference guide.

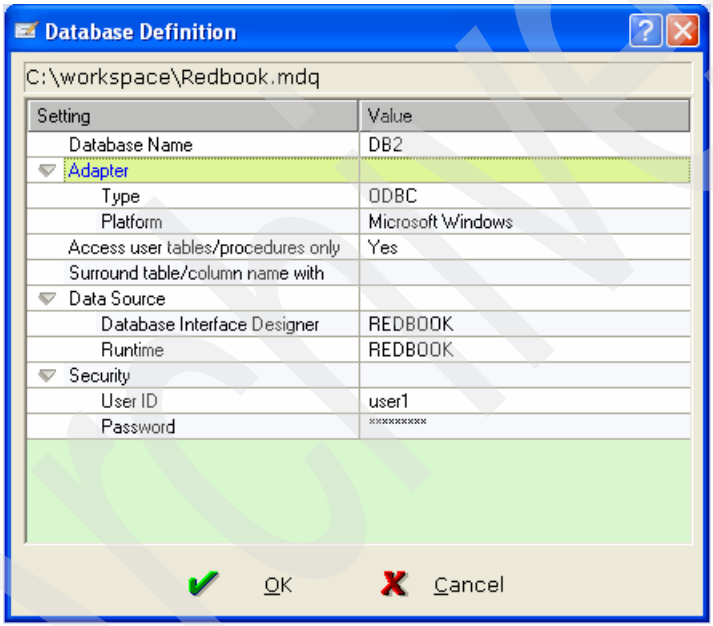

Figure 5-134 Completed Database Definition window

In the Database Definition window, you can specify values for the following settings:

- The *Database Name* is the name of the database that is being defined. This name is displayed in the Navigator.
- For the *Adapter Type*, you select from a list of adapters that can host the database that you are defining. Select one from the list. The default value is

ODBC. This selection affects the list of supported platforms that is displayed in the Platform list.

- The *Adapter Platform* is a list of platforms upon which the adapter (that you selected in the Type list) is supported.
- The *Access user tables/procedures only* setting determines the level of user access and, to some extent, the presentation of the information that is being presented. The default value is Yes.
  - If No is selected, all of the names (of tables, views, and so on) in the database are accessible and are presented, including the respective prefix of each associated user ID.
  - If Yes is selected, only the names (of tables, views, and so on) in the databases associated with the current user ID are accessed and presented.

  Because the names are all associated with the same user, no prefix is added.
- The *Surround table/column names with* setting indicates the character that you want to use to enclose table and column names. For example, many databases require that names be enclosed by single or double quotation marks if they have spaces in their names, or when the names are SQL reserved words. When you specify table names in output cards or column names in update keys, these names are enclosed by the character that is defined in this setting to provide database compatibility. There is no default value.
- The *Data Source - Database Interface Designer* and the *Data Source - Runtime* setting is platform-specific. On other platforms, you might have different settings.
  - The *Data Source - Database Interface Designer* setting is the name of the data source used to access database information for design-time (pre-production or testing) purposes.
  - The *Data Source - Runtime* setting is the name of the data source used to access database information for run-time (map execution) purposes.

  Again, see the platform-specific adapter reference guide or the context-sensitive help for more setting-specific information.
- The *Security - User ID* setting is the user ID that is used to access the database. There is no default value.
- The *Security - Password* setting is the password that is used to access the database. There is no default value.

The new database name is displayed in the Navigator next to the Databases icon (Figure 5-135).

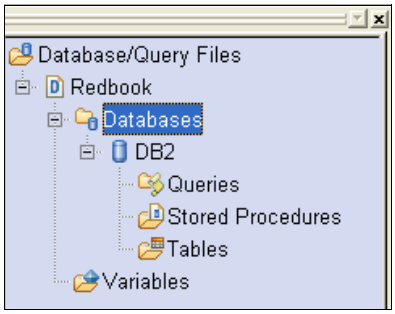

Figure 5-135 New database defined

After you open a .mdq file in the Database Interface Designer window and a define a database, you can specify queries. To use a query as a source, define the query by assigning it a name and entering either the SQL SELECT statement or the stored procedure invocation statement.

To define a query:

1. In the Navigator, select **Queries** under the database icon for which you want to add a query.
2. From the Query menu, select **New**.
3. In the New Query window (Figure 5-136 on page 304):
   a. In the Name field, enter a name for your query. Use this name to reference this query by using the Map Editor, or use it in a data source override execution command.
   b. In the Query text box, enter an SQL statement that defines how data should be retrieved from the database. If your SELECT statement includes table names with spaces, you *must* enclose them with either single or double quotation marks.
   c. Click **OK**.

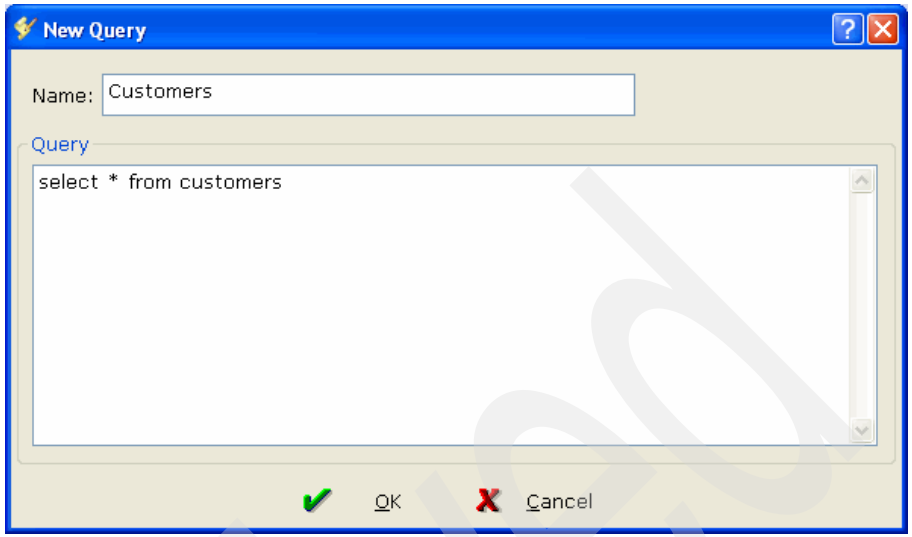

Figure 5-136 New Query window

The new query is displayed in the Navigator next to the query icon (Figure 5-137).

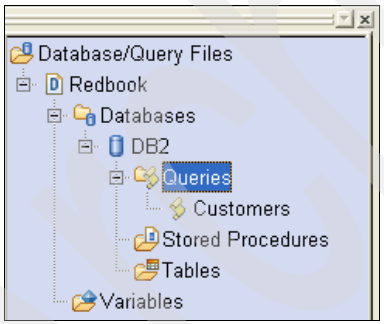

Figure 5-137 New query defined

Defining variables in SQL statements

Elements of SQL statements can be executed as map sources that are determined at run time. You can use the Database Interface Designer to define a statement variable with a pseudo value in an SQL statement and then pass the actual value on the command line at run time. This technique is beneficial when using the RUN function because it allows one map to modify the SQL statement of another map or to build, potentially, an entire SQL statement.

When you generate type trees by using the Database Interface Designer, you must enter a substitution value for each variable to ensure that the syntax of the SQL statement is valid. The Database Interface Designer provides a facility for specifying a value for these variables. However, the value that you enter for a variable in the Database Interface Designer does not have to be the same value that is passed at run time. Any value can be passed.

In the Database Interface Designer Query window, variables can be specified in SQL statements as literals enclosed in the number sign (#) character. For example, you might enter a statement that defines a variable named ID as in the following example:

```
select * from BigTable where Identifier = #ID#
```

Because the value of a variable can be a text string, you can also create larger elements of the statement variable as in the following example:

```
select * from BigTable where #WhereClause#
```

When you define variables in a query, the Database Interface Designer automatically detects the presence of the variables in the statement and lists each variable in the Navigator, along with a variable icon. Use the Define Variables window to enter the variable values.

For example, you edit the query from Figure 5-136 on page 304 to add a “where” clause with a variable (Figure 5-138).

Figure 5-138 Query with a variable

After you click OK, the variable is known and displayed as shown in Figure 5-139.

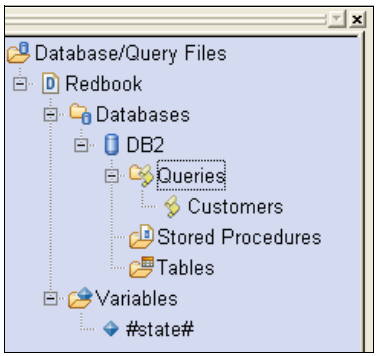

Figure 5-139 New variable in Navigator

Variables: You cannot generate a type tree for the query until you specify a value for each variable that it contains. Also, if the variable that you are defining in the Define Variables dialog is a text string, you must enclose the value in single quotation marks.

The pseudo values that you specify by using the Define Variables window are used in the Database Interface Designer only when accessing the database to generate the type tree. They are not used when executing a map.

To specify values in the Define Variables window:

1. In the Navigator, select the variable to which you want to add one or more values.
2. From the Query menu, select **Define Variables** (Figure 5-140).

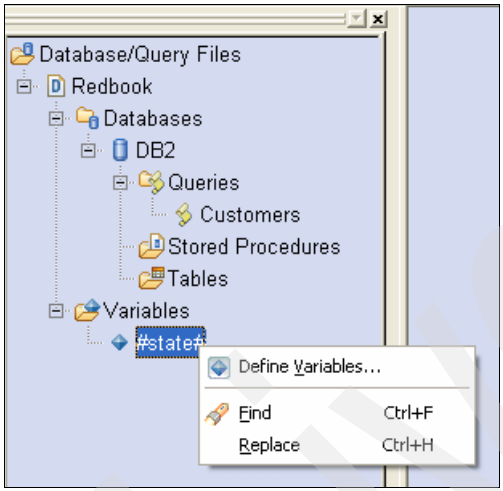

Figure 5-140 Define variable menu option

3. In the Define Variables window (Figure 5-141), which lists all the variables in the .mdq file, in the Value field, enter a value for each variable. In the example, we enter the `NY` value. Then click **OK**.

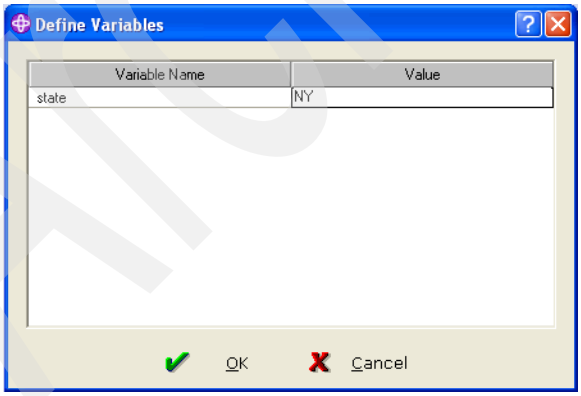

Figure 5-141 Adding the variable value

The pseudo values are stored in the .mdq file and can only be changed in the Define Variables window. Also, if the variable that you are defining is a text string, you must enclose the value in single quotation marks.

Even though you have specified pseudo values for the defined variables by using the Database Interface Designer, the values are only substitution values that are not referenced at run time. Values must be provided for the variables at run time. Otherwise, the SQL statement is syntactically invalid.

The correct value for each variable is specified at run time by using the Variable adapter command (**-VAR**) in the Input Source - Override execution command (**-ID**).

Generating type trees

Use the Database Interface Designer to generate a type tree from a table, query, stored procedure, or message queue that is associated with a particular database. A type tree that is generated by using the Database Interface Designer contains a type for the table, query, stored procedure, or message, as well as a row type and types for each column in that table, query, stored procedure, or message. As an alternative, use the **mtsmaker** command on nonWindows-based platforms to generate type trees when you cannot use the Database Interface Designer.

When you generate a type tree for a table or view, information from your database and from the Database Interface Designer is used to create a type tree file. If you change your database table definition or edit the database definition in the Database Interface Designer, you must generate a new type tree to reflect the new information. Generated type trees can differ when generated by using different database adapters. For example, a date field might be represented differently by different databases. For this reason, use the same adapter type that will be used at run time to generate the type tree with the Database Interface Designer.

To define the type tree to generate from a table or view, use the Generate Type Tree from Tables window or the Generate Type Tree from Views window.

To generate a type tree from a table or view:

1. In the Navigator, highlight the icon of the database that contains the table from which you want to generate a type tree.
2. From the Database menu, select **Generate Tree From → Table**, or in the Navigator, right-click the database icon and select **Generate Tree From → Table** (Figure 5-142).

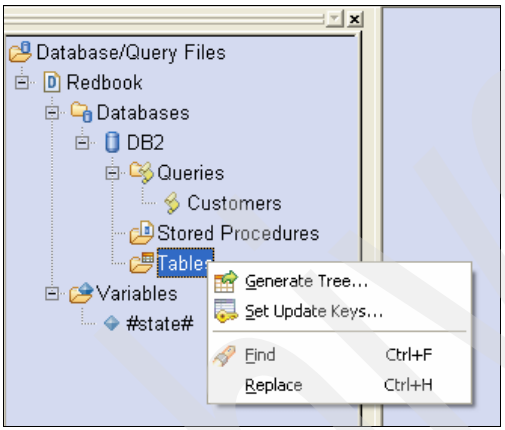

Figure 5-142 Selecting the Generate Tree option

3. In the Generate Type Tree from Tables window (Figure 5-143), which includes a list of tables and views:

   a. From the Tables/Views list, either select or add a table:

      - To select one table in the Tables/Views list, then click the table. To select multiple tables, press either the Shift key or Ctrl key and click the desired tables.
      - If the table or view that you want is not displayed in this list, add it by right-clicking in the list, selecting **Insert**, and typing the desired name.

   b. Create a new type tree. In the File name field, specify the path and name (with a .mtt extension), or click the **Browse** button. In the Save As window, highlight it and click **OK**.

   c. Specify the remaining options as desired. Refer to the context-sensitive help for field-specific information.

   d. Click **Generate**.

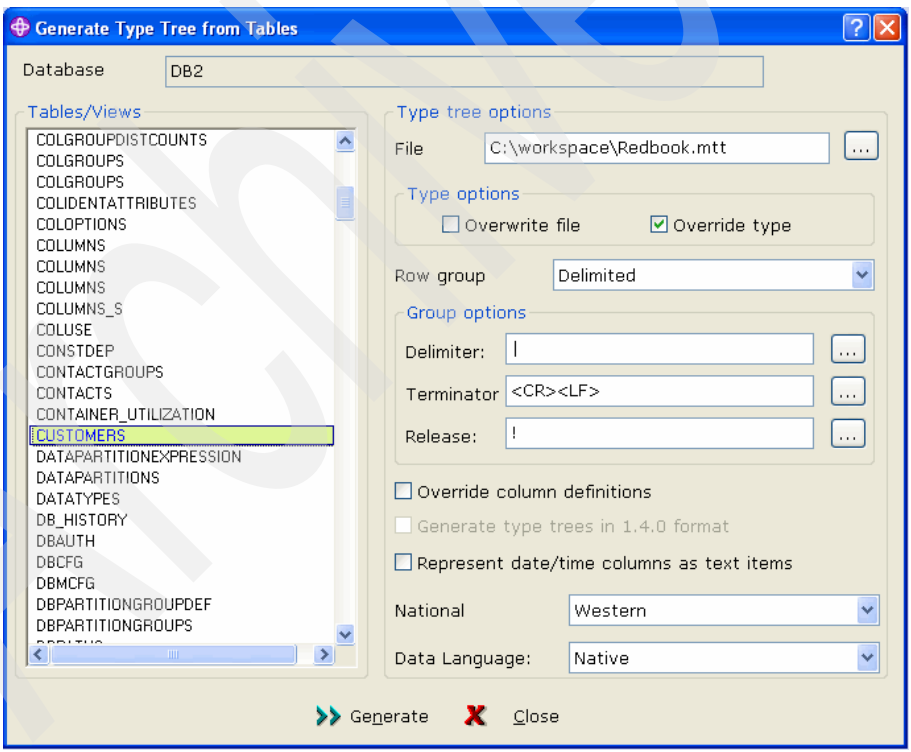

Figure 5-143 Generate Type Tree from Tables window

The Database Interface Designer and the Type Tree Maker produce a type tree that corresponds to the selected database tables or views (Figure 5-144).

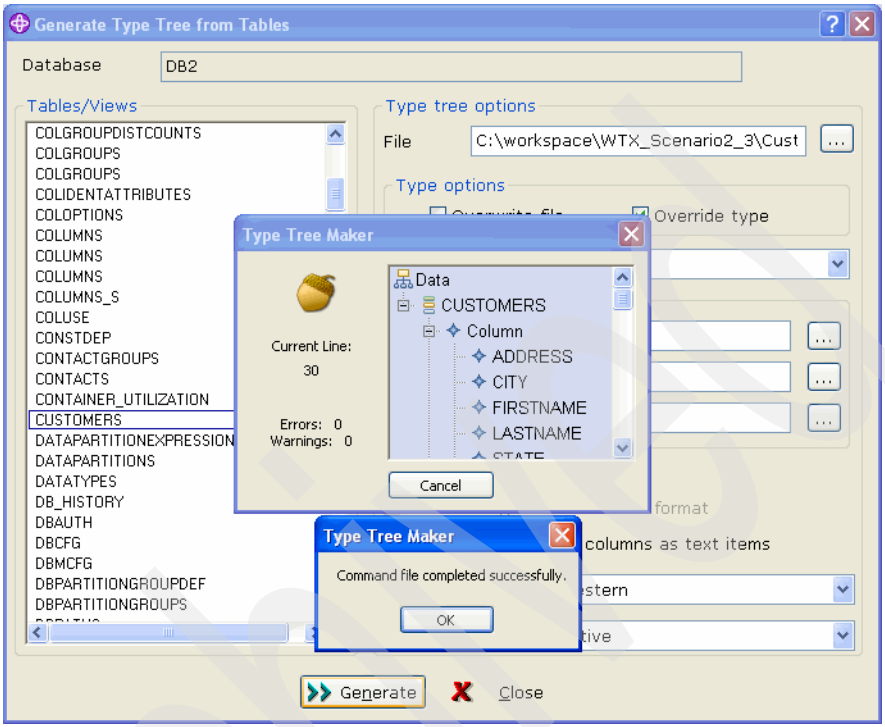

Figure 5-144 Type Tree Maker generating a type tree

The generated type tree is represented in the Navigator with the table/view icon next to the name (Figure 5-145).

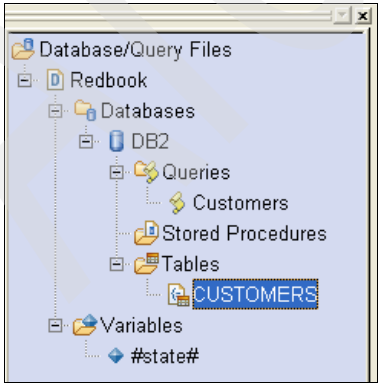

Figure 5-145 Navigator with a type tree generated from a table

Figure 5-146 shows the generated type tree opened in the Type Tree Editor.

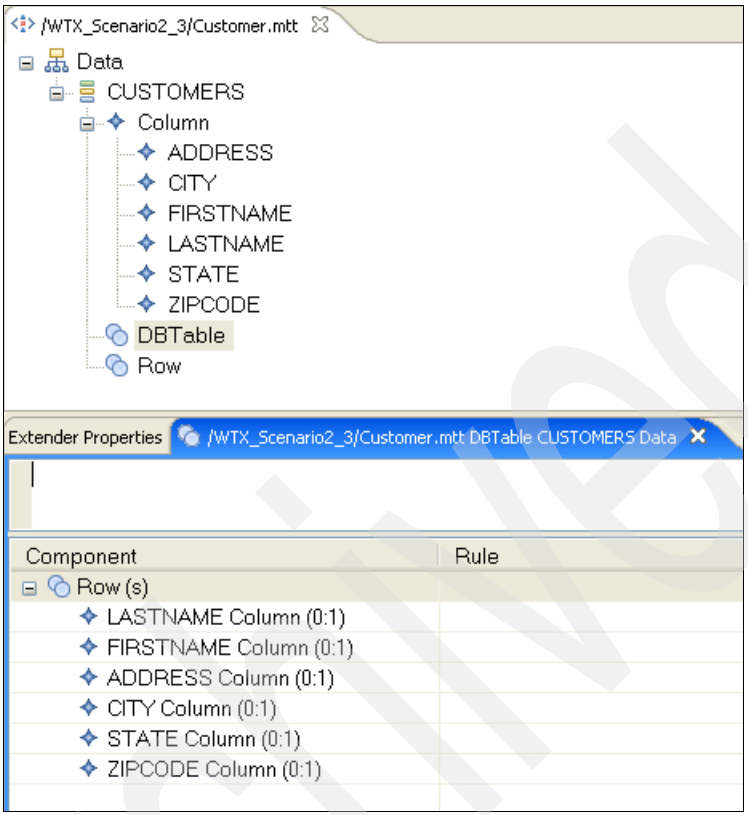

Figure 5-146 The generated type tree

You can also generate a type tree from a stored procedure, a query, or a message queue. Consult the documentation for details about the differences and steps to generate the type tree.

After you generate a type tree, the type tree can be used in a map in an input card to read records or in an output card to update records. For an input card, you can use a type tree based on either a standard SQL SELECT query or a query by using a stored procedure. You can use a type tree based on a database table, view, message queue, or stored procedure as a target. See 5.3.11, “Adapters: The GET and PUT functions” on page 330, for more detail about using adapters in map cards.

To illustrate creating and using a database/query file, we present the following example.

In our scenario in Chapter 8, “Integration scenarios: WebSphere Transformation Extender for Integration Servers” on page 475, we create a map named MT103chkamt. (See 8.6, “Creating the second WebSphere Transformation Extender map” on page 582.) The map has an output card (card #2) for which we create a type tree by using the Database Interface Designer. In addition, we have a DB2 Table named MT103 (Figure 5-147) that is the target location of our alert.

Table - MT103

Schema : WTXADMIN
Creator : WTXADMIN
Columns : 9

Columns

Key	Name	Data type	Length	Nullable
	TRANS_ID	BIGINT	8	No
	SENDER_BIC	VARCHAR	30	Yes
	RECEIVER_BIC	VARCHAR	30	Yes
	CURRENCY	CHARACTER	3	Yes
	AMOUNT	DECIMAL	30	Yes
	AMOUNT_EUR	DECIMAL	30	Yes
	STATUS	VARCHAR	50	Yes
	BIC_RETURN_CODE	VARCHAR	50	Yes
	BIC_RETURN_MSG	VARCHAR	50	Yes

Figure 5-147 DB2 table used as the format for the alert file

As mentioned previously, you can use a type tree based on either a standard SQL SELECT query or a query by using a stored procedure. The following steps illustrate how to create and use a database or query file:

1. Start the Database Interface Designer. In the window that opens, select **Create a new database/query file**.
2. In Design Studio (Figure 5-148 on page 314), click the **Save** icon or press Ctrl+S. Change the default name from `Database_QueryFile1.mdq`, and the directory to your project directory with a better logical name (such as `swift.mdq`). This file can contain all database elements for your project.

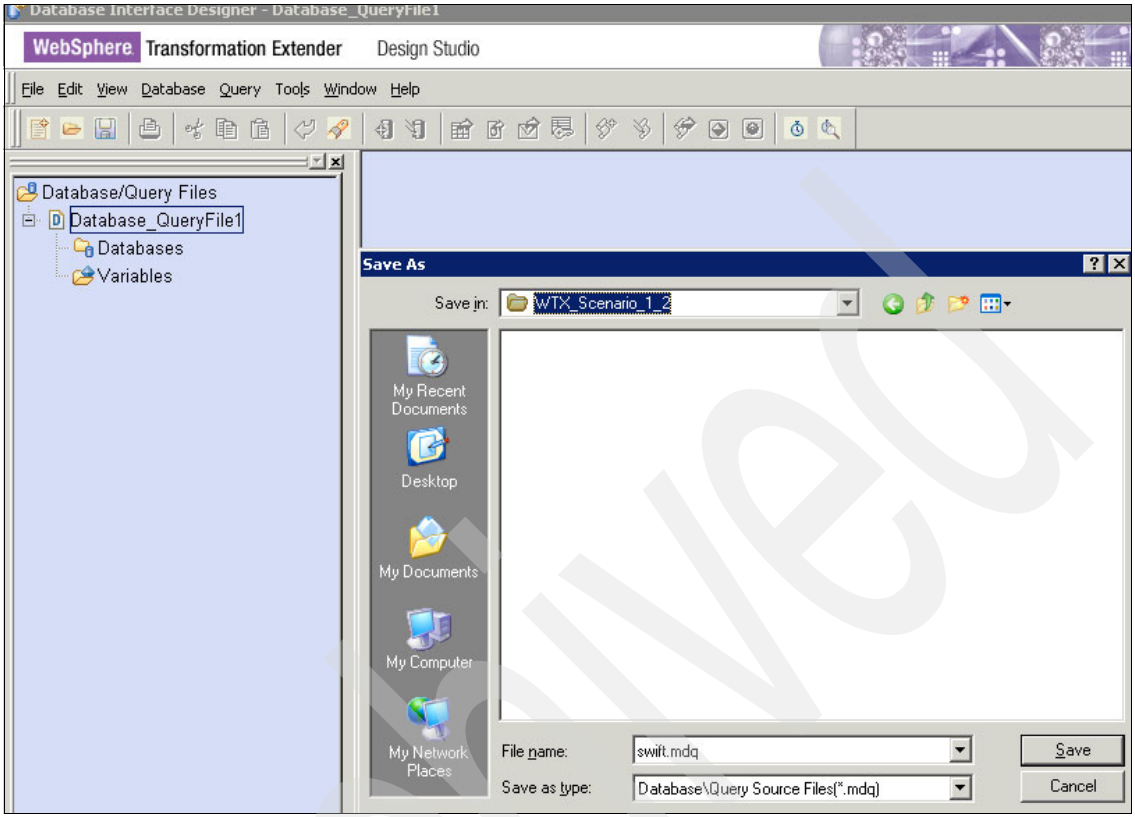

Figure 5-148 Creating a new Database Query file in our project directory

3. Press Ctrl+D to create a new database entry.

4. In the Database Definition window (Figure 5-149):
   a. Specify the logical database name of `DB2`.
   b. Enter the Adapter Type, which is `DB2` in this example, and the Platform of the database, which is `Windows` in this example.
   c. For Access user tables/procedures only, the default value here is `Yes`. For this example, keep the default value.
   d. Under DataSource, select the database to be used during development (during type tree generations and so on) and run time (map execution). The values depend on the adapter type that you selected earlier. By using a different database for development and runtime execution, you can reduce access on the runtime dataserver and use a mirror database instead.
   e. For User ID and Password, enter the same values that you defined for WebSphere Transformation Extender (with all the necessary rights). If a user ID and password are not specified, the current user ID and password are used for the user of the operating system session that is running.

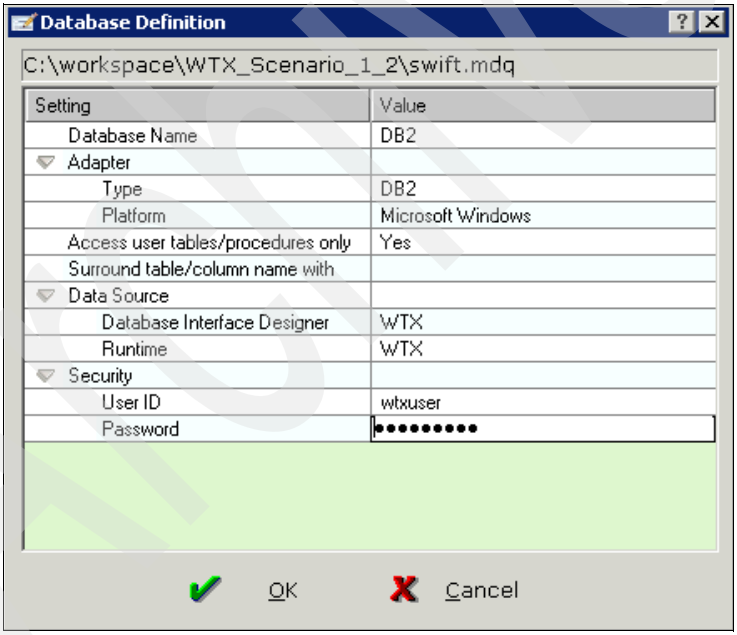

Figure 5-149 New database definition in the swift.mdq file

5. Test the new database connectivity and generate the type tree:

   a. Click the **Generate tree from table** icon.

      If a window does not open, activate the trace and see the database log file to identify the problem.

   b. Select the **MT103 table**, rename the type tree file to `MT103_TABLE.mtt`, and click **Generate** to invoke the type tree maker and generate the appropriate type tree. The top level element is DBTable.

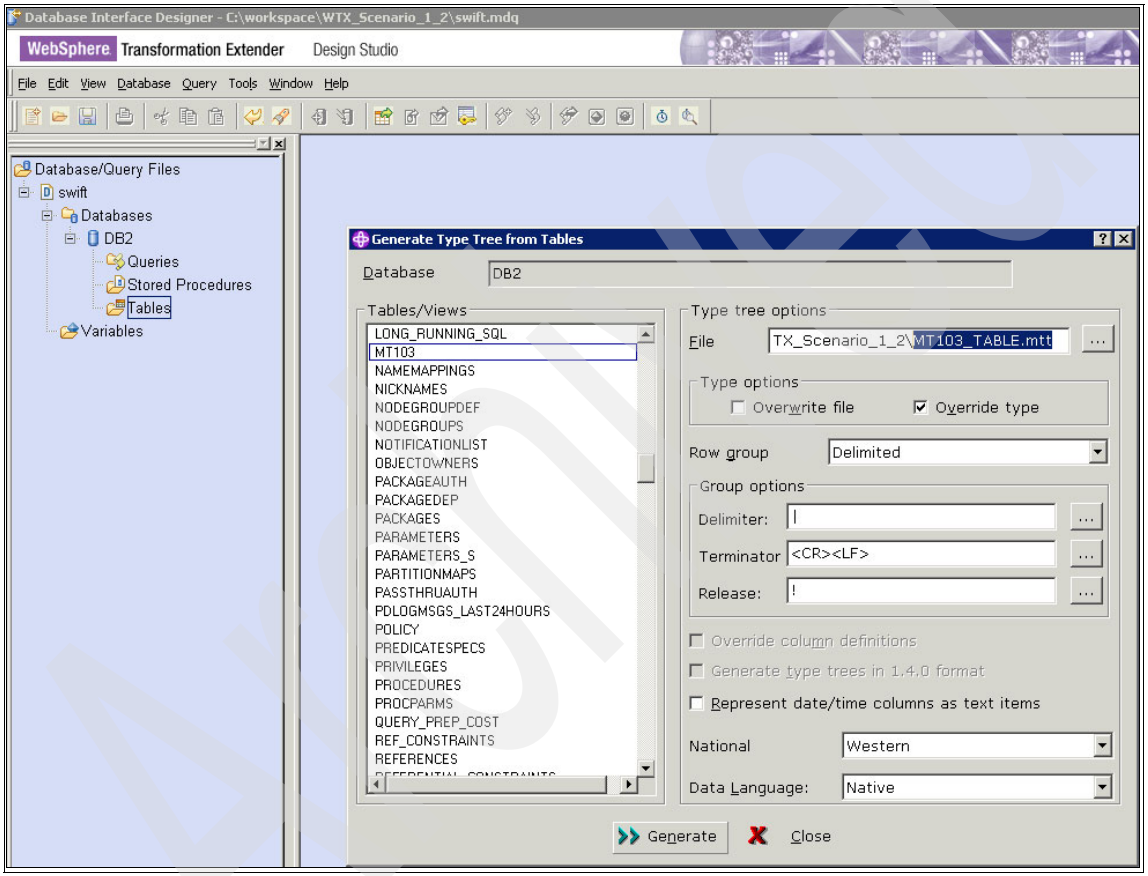

Figure 5-150 Generating MT103_TABLE.mtt type tree from MT103 table

6. Use the generated type tree in a map:
   a. Go back to Design Studio to see the `swift.mdq` database query file and the `MT103_TABLE.mtt` type tree file:
      i. Select the project in the Extender Navigator view.
      ii. Right-click and select **Refresh** (or press the F5 key) to display the type tree and the database query files.
   b. Add an output card to a map.
   c. Type the card by using **DBTable MT103** data from the `MT103_TABLE.mtt` file.
   d. Use the database adapter and database query file `swift.mdq` file path, Database **DB2** (from dropdown menu) and Table **MT103** (manual insert of the table name)
   e. During the development period, you may want to add the "-T" adapter command to generate a database adapter trace file (**MT103chkamt.dbl**) in case of database issues.

Alternate method to the database query file: You do not need a database query file to connect to the database. An alternate method is to add the following command on the adapter command line:

```
-TRACE -DBTYPE DB2 -SOURCE WTX -TABLE MT103 -USER wtxuser
-PASSWORD wtxuser
```

As an alternative, you can use the short form of the command:

```
-T -DT DB2 -DS WTX -TB MT103 -US wtxuser -PW wtxuser
```

However, this command is not recommended for a production implementation. Anyone who opens the map can see the database user ID and password. It is hardcoded in the map, which can cause maintenance issues. An alternative is to use the resource registry, but again security is not optimized then. Although a resource can be encrypted, anyone who has the resource registry tool installed can easily open the resource naming file and see, in decrypted format, the user ID and password.

A database query file is one possibility for using passwords confidentially. The password can be inserted by the database administrator so that no one can easily access it.

7. Enter map rules to map the source to the (database) target.
8. Build and run the map.

9. As long as the data structure does not change, overwrite the settings of this output card to generate data to any application. By using the command server, execute the map and select where you want to generate the data. For example, type the command in Example 5-6 to generate the second output card into the alert_file.txt file inside C:\$Redbook\WMB_Scenario\ouput_maps.

Example 5-6 Generating the second output card into the alert_file.txt file

```
C:\IBM\WTX8.2\dtxcmdsv.exe<SP>C:\workspace\WTX_Scenario_1_2\
MT103chkamt.mmc<SP>-OF2<SP>C:\$Redbook\WMB_Scenario\ouput_maps\
alert_file.txt
```

5.3.9 Unit testing

You can run a map to unit test it from within the Map Editor, after the map has been built successfully (for Windows).

To run and test a map:

1. With the map open, select **Map** → **Run**.
2. In the Command Server window, review the progress of the map execution, the total number of input objects found, and the total number of output objects built. These statistics are shown to indicate progress as the data is mapped.
3. If the map executes successfully, in the message box that opens (Figure 5-151), click **Cancel** to close the box.

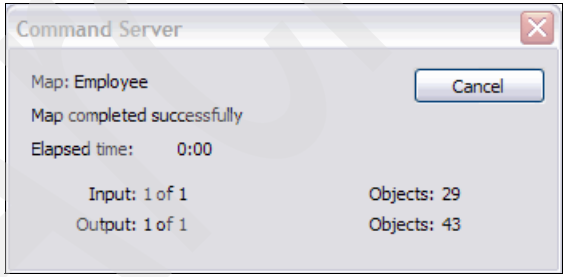

Figure 5-151 Successful map run

4. View the results of the map execution from within Design Studio.
   a. Click **Map** → **View Run Results**.
   b. In the window that opens, select or clear the input(s) and output(s) that you want to review (Figure 5-152) and click **OK**.

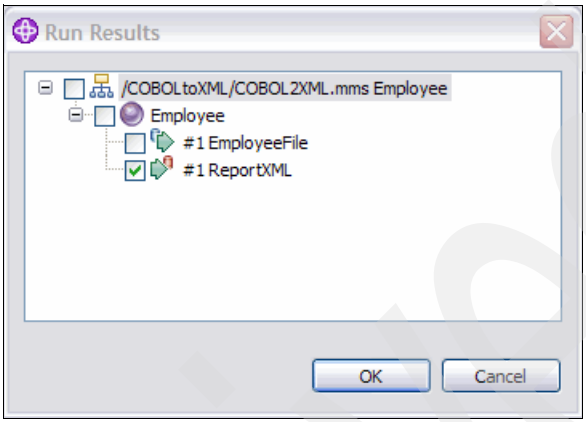

Figure 5-152 Run Results window

A new tab opens in the Design Studio for each file that is displayed. You can review the data to verify if your mapping rules are generating the expected output. Then close the tab when you are done.

If the map does not run successfully, you see a response as in the example in Figure 5-153. In this case, there are no run results to view. At this point, you must trace the map to determine the source of the errors.

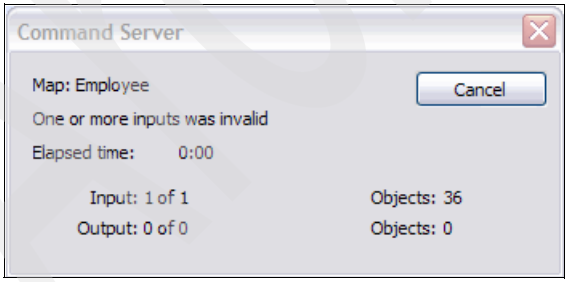

Figure 5-153 Unsuccessful map run

5.3.10 Tracing and debugging

Various troubleshooting methods can assist you in situations where the following types of errors occur regarding input or output data:

- Data is invalid.
- Data contains invalid objects.
- Unknown data exists.

When you run a map, first the input data is validated by comparing the data to its definition in the type tree. After validating data, the output card map rules are evaluated, and the output objects are built.

If you run a map and invalid or unknown input data is encountered, you can analyze what happened during the validation process. In addition, you can analyze the output data.

After a map is run, a runtime message is displayed in the Command Server window. This message provides information about the map. For example, the message "`Map completed successfully`" indicates that all inputs are valid. However, you might see the message "`One or more inputs was invalid`." This means that an input card object is invalid, as in Figure 5-153 on page 319.

The message "`One or more inputs was invalid`" can occur even if you are absolutely sure that the input data is valid. In this case, there might be an error in the data definitions in the Type Tree Editor. For example, if you mistakenly defined an item to be an integer but the data is actually a decimal number, that data object is considered invalid because it does not match its type definition. In general, when data is diagnosed as "invalid," either the data is invalid or the type definition is incorrect.

The Command Server window also shows the progress of the data that is mapped. For example, you can see when input objects are found and when output objects are built.

Review the run results of your map execution to see the output data that is generated by the map. The run results are the actual data as described by the data objects that are defined in the input and output cards.

By viewing the results of a map execution, you can gain important information about how the data was mapped. For example, when no output objects are built, your input data might be invalid. If you see only a portion of the output data, your input data might be valid, but it can contain unknown data.

You might want to audit the data, as explained in "MapAudit settings" on page 272. The audit log stores the information about your data that you specify

from within the map settings. For example, you can specify that the audit log only records failure information.

Another way to troubleshoot a map is to customize the validation settings. Validation can be set to ignore certain aspects of the data during validation by setting custom options in the Validation map settings. For example, you can have the validation process ignore item restrictions. This means that, even if an item has restrictions defined, the restrictions are not considered when that item is validated.

Tracing

A *trace file* is a text file that records map execution progress when enabled. You can enable the trace option from the map settings or from the adapter command line. You can use the information in the trace file to diagnose invalid data or incorrect type definitions.

You can choose to trace the input data, the output data, or both. When input data is traced, the trace file provides a step-by-step account of the data objects that are found, the reason the data is found to be invalid, the sizes and counts of the data objects, and the data object position in the data stream. When output data is traced, the trace file specifies the objects that are built and the output objects that evaluate to “none.”

Important: Because performance can be impacted, enable the trace option only during debugging.

For input data tracing, a message exists in the trace file for each input data object indicating whether the data object is valid. If the data is invalid, the message indicates the reason that the data is invalid.

The name and location of the trace file are defined in the Map Settings window. The default name of the trace file is the executable map name, plus the .mtr file name extension. For example, an executable map with the `MyMap` name has a trace file with the `MyMap.mtr` name. The default location of the trace file is the map execution directory.

If you ran a map and received a message that an input was invalid, turn on the input trace only. If you received a message that an output was invalid, turn on the output trace only. If you trace both input and output, the input messages are at the beginning of the file, followed by the output messages.

You have three options for tracing that you set in the Map Settings window (Figure 5-154). The *SummaryContentTrace* option can be turned ON or OFF to determine the card that failed. The *InputContentTrace* option traces on input. The *RulesTrace* option traces on output. A master Switch setting controls whether tracing is enabled (ON) or not enabled (OFF).

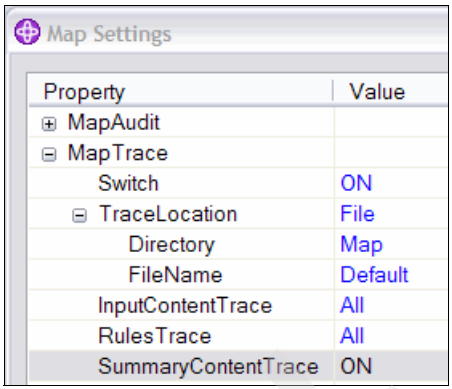

Figure 5-154 MapTrace settings

Note: The map settings are saved with the map. Therefore, if you alter the trace settings for a map, you must build the map again.

After you run a map, you can view the trace file in the Trace view. If the Trace view is not visible, you can display it from Design Studio by selecting **Window** → **Show View** → **Trace** (Figure 5-155).

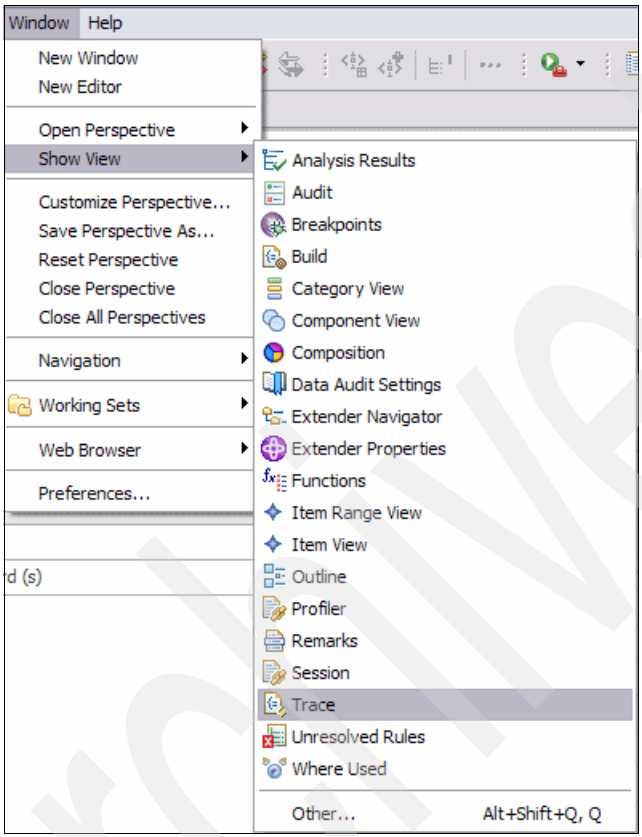

Figure 5-155 Show View menu

A trace file of input messages contains a message for each input data object. A message gives the following information:

- A date and time stamp
- Description of the level of the object in the card
- The offset of the data object in the data stream
- The length of the data object in bytes
- The component number of the data object
- The index of the component
- A portion of the actual data
- The name of the type to which it presumably belongs

Figure 5-156 shows a portion of a trace view. In the trace file, the symbol \r indicates a carriage return. The symbol \n indicates a line feed. The DI number and X number are not used for debugging purposes and can be ignored.

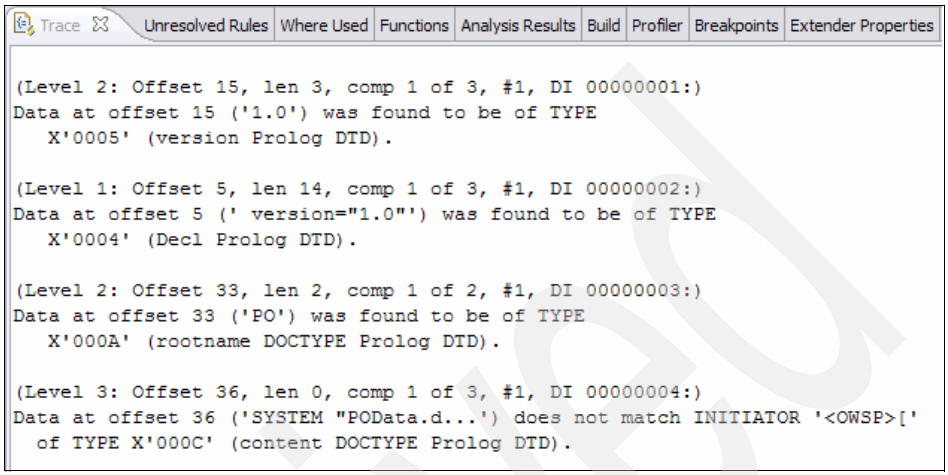

Figure 5-156 Sample trace file

To explain the trace output further, review the example in Figure 5-157, which shows a trace message for the EMP_LAST_NAME object. The length (len) of the data is 15 bytes. The level of an object indicates the position of the data object with respect to the entire card object. A card object has level 0. A component of the card object has level 1, and so on.

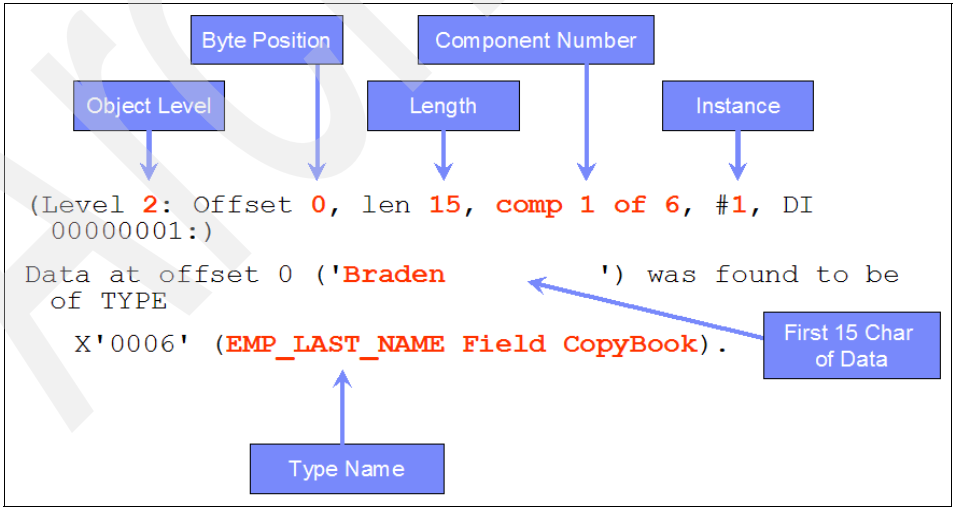

Figure 5-157 Details of the trace file example

When a data object does not match the definition of the type to which it presumably belongs, it might be invalid. An invalid data object might occur for the following reasons:

- An item value is not in a restriction list.
- An item failed the presentation test.
- An item is the wrong size.
- A data object has the wrong delimiter.
- A data object has the wrong terminator.
- A data object failed the component rule.
- A group is missing a required component.
- A group contains one or more invalid component or components.

To try to quickly find the source of the issue, use the tips in Table 5-1.

Table 5-1 Tips for determining the source of an invalid data object

If you see this runtime message:	**Limit trace to:**	**Read this in the trace file:**
One or more inputs was invalid	The summary	Read the card summaries to determine which card is invalid.
	The invalid card	Start searching from the bottom, and look for the word *invalid* in the trace file.
Input valid, but unknown data found	The summary	Read the card summaries to determine which card contains the unknown data.
	The card that contains the unknown data	Start searching from the bottom, and look for the word *invalid* in the trace file.

Debugging

The debug functionality in the Map Editor can assist you in creating and troubleshooting maps. Use the Map Debugger after you narrow down a problem to a certain object or function call, such as when you know that an object is not producing the correct output.

With the map debug functionality, you can break the execution of a map and inspect the current object values (both input and output objects) that are used within the currently executing output rule.

Important: You cannot use the Map Debugger when the Map Profiler is enabled.

By enabling the Map Debugger, you perform the following tasks:

- Identify breakpoints.
- Step through rule execution.
- View object values at a rule level.

To enable map debugging, click the **Debug Map** icon on the toolbar (Figure 5-158).

Figure 5-158 Debug Map icon

After the Map Debugger is enabled, the icon changes to show a white box around it (Figure 5-159).

Figure 5-159 Debug Map icon with debugging enabled

You can set breakpoints on the output objects of functional and executable maps. You can only add a breakpoint when the Map Debugger is enabled. To add a breakpoint, from an output card, right-click an output object and select **Add BreakPoint** (Figure 5-160).

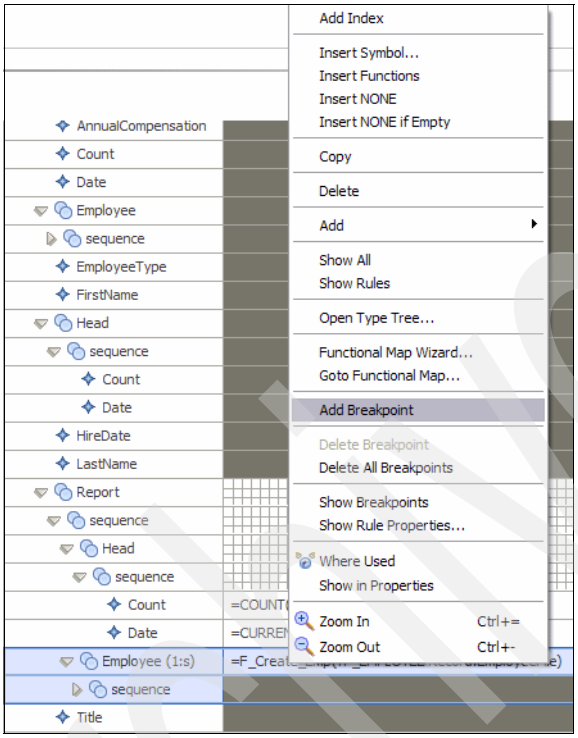

Figure 5-160 Adding a breakpoint

The output object rule with the breakpoint becomes highlighted (Figure 5-161).

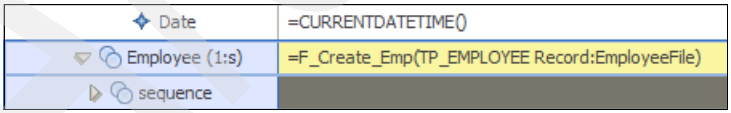

Figure 5-161 Highlighted rule indicating a set breakpoint

To delete a breakpoint, right-click the output object and select **Delete BreakPoint**. You also have the option to select **Delete All Breakpoints**.

When you save a map after adding breakpoints, the breakpoints are also saved. Therefore, the next time you open the map and enable the debugging option, the saved breakpoints are re-displayed.

To see the breakpoints that you set in a map, right-click the output object and select **Show Breakpoints**, which opens the Breakpoints view (Figure 5-162).

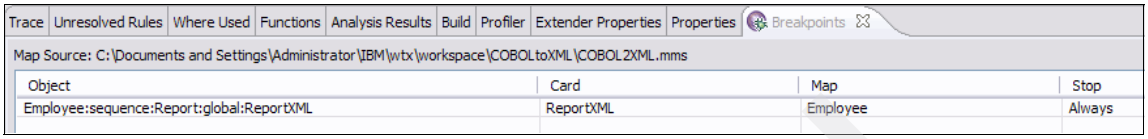

Figure 5-162 Breakpoints view

You can configure the Map Debugger to stop on a specific iteration of a rule. For example, a rule repeats 10 times and the ninth occurrence is wrong. You can configure the Map Debugger to stop on the ninth iteration and subsequent iterations of the rule.

To set a conditional breakpoint:

1. Right-click a card object and select **Show BreakPoints** (Figure 5-162).
2. In the BreakPoints view, select an object from the list, and click in the **Stop** column.
3. In the Stop column, click the ellipsis (...) button (Figure 5-163).

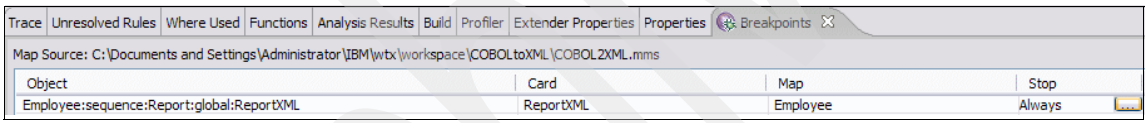

Figure 5-163 Setting a conditional breakpoint

4. In the Breakpoint Condition window (Figure 5-164):
   a. In the Condition field, select **On Iteration**.
   b. In the Value field, enter a numeric value that represents the iteration of the rule where you want the Map Debugger to begin stopping. For example, type 9 for the Map Debugger to begin stopping on the ninth iteration of the rule.
   c. Click **OK**.

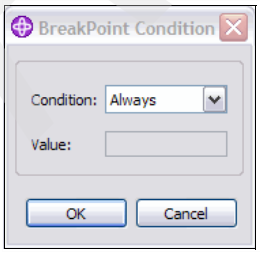

Figure 5-164 Setting the breakpoint condition

The Breakpoints view is updated with the modification (Figure 5-165).

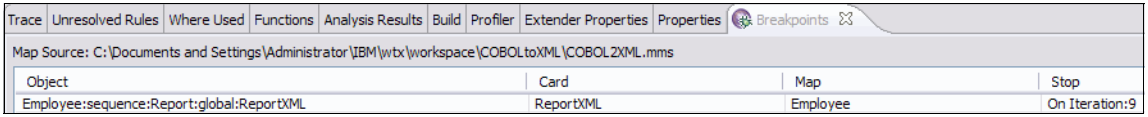

Figure 5-165 A conditional breakpoint

When the map runs in this case, the Map Debugger stops at the ninth iteration of the rule and then on every occurrence of the rule thereafter.

After you have the desired breakpoints set, build and run the map. Map execution pauses when it encounters the object for which the first break was positioned. A debug window opens (Figure 5-166).

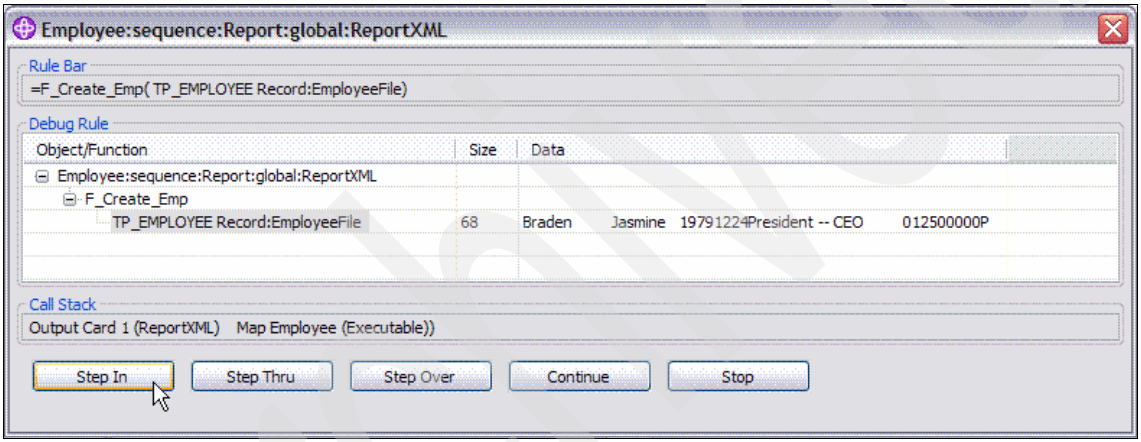

Figure 5-166 Debug window

Starting from the first breakpoint, the debugging process can step through every rule on every card. The debugging process can step into functional maps and follow RUN maps that are defined within the same map source file, provided that the maps have been built with the Debug option turned on.

If any referenced functional maps or external RUN maps are found, the map enters step-over mode and stops at the end of the originating rule.

When processing a rule, the debugger displays relevant data. Relevant data is data that is read and written in the rule. The data that is currently being processed by a rule or function is displayed in the Debug Rule window.

The rule bar in the Debug Rule window shows the current rule in a format of a hierarchical tree of objects, functions, and maps. At each breakpoint at which the map pauses, input and output values of objects, functions, and rules are shown.

At this point, you can choose from the following options:

- *Step In* to execute the next object or function in the rule

  For objects, the current value is displayed. For functions, the output displays the result of the particular function.

  - At a rule level, the debug process proceeds to the next function or returns the value of the next object in the current rule.
  - At the function level, the object or map evaluated is updated, displaying the data that was returned.
  - At the object level, the value of the object is displayed.
  - At the function and map level, the result of the particular operation is displayed.

- *Step Thru* to step through the rule until you recognize the problem

  When the current rule executes, the results are displayed. At a function level, this option does not debug the remainder of the functions or objects in the rule.

- *Step Over* to move to the next rule in the current map
- *Continue* to continue map execution and stop at the next breakpoint if one exists

  Otherwise continue to the end of the map.

- *Stop* to stop map execution

  Execution ceases when all RUN maps have completed.

5.3.11 Adapters: The GET and PUT functions

WebSphere Transformation Extender resource adapters are used to retrieve and route data. They provide access to databases, files, messaging systems, enterprise applications, and other data sources and targets. You can use adapters with a map in a map rule and with the Command Server, Launcher, Software Development Kit, and so on.

When using adapters, several card settings are available that you must review and set.

Input cards

The Retry settings (under the GET property of an input card) specify the number of times, and at what interval, to attempt to access an unavailable source (Figure 5-167). Set Switch to ON (enable) or OFF (disable). MaxAttempts is the maximum number of attempts that are made to access that source. The Interval is the time to wait in seconds before making the next attempt to access that source.

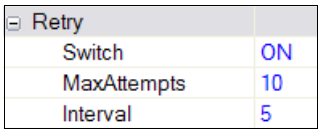

Figure 5-167 Adapter Retry settings

The Transaction settings (under the GET property of an input card) identify when to apply OnSuccess and OnFailure actions (Figure 5-168). Applying the OnSuccess and OnFailure action is based on the success or failure of the following items:

- An entire map (a map = one or more bursts)
- Each burst of data
- The card

The availability of map, card, and burst scope options is determined by the adapter type.

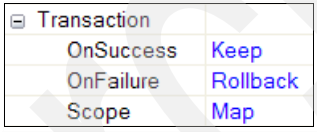

Figure 5-168 Adapter Transaction settings

The OnSuccess and OnFailure settings work in conjunction with the Scope setting. The OnSuccess setting has the following options:

Keep Do not remove data from its source.

KeepOnContent Do not remove input data from its source, unless it has no content.

Delete Delete or remove data from its source.

The OnFailure setting has the following options:

Rollback Rollback (undo) changes.

Commit Commit changes.

Output cards

Output card settings identify actions to take when the map runs, such as what data to send, where to send the data, and what to do with the data when the map completes. Setting and value options vary by adapter. Most settings can be overridden at run time. The Backup, OnFailure, Retry, and Scope settings are the same for both input cards and output cards.

The OnSuccess setting has the following options for an output card:

Create	Send data to the target.
CreateOnContent	Send data to the target only if there is content.
!Create	Regardless of whether the data content is produced, do not send data to the target.
Append	Append data to an existing data file.

GET and PUT functions

You can use the GET function to retrieve data by using one of the source adapters, such as a messaging system, a database, a file, and so on, within the course of your map. When the GET function is used in a map, the default OnSuccess action is adapter specific. The default OnFailure action is to roll back any changes that are made during map processing. The default Scope is integral unless the map is defined to run in bursts, which is the case when one or more inputs have the FetchAs property set to Burst.

The PUT function passes data to a target adapter. Use PUT to route data within your map by using one of the target adapters such as to a messaging system, a database, a file, and so on.

When the PUT function is used in a map, the default OnSuccess action is adapter specific. The default OnFailure action is to roll back any changes that are made during map processing. The default Scope is integral unless the map is defined to run in bursts, which is the case when one or more inputs have the FetchAs property set to Burst.

5.3.12 Resource Registry

The Resource Registry is an application to define and manage the resource name aliases, which are specified in map cards and maps or systems settings. It provides an alias repository that can be used to abstract various parameter settings by using aliases that resolve to specific resources within the enterprise.

The premise behind the Resource Registry is that it permits generic resource variables to be used within WebSphere Transformation Extender components.

These resource variables are then automatically resolved to the actual resources, such as directory paths, database names, message queue names, and server names. Resource variables make it possible to use the same map and system definitions in different deployment environments.

Basic concepts

A *resource variable* is an alias that can have different values that are defined for multiple deployment environments. An appropriate value of the resource variable is resolved during the map execution time.

A resource variable can be used in the following situations:

- In the input/output card adapter settings
- In the input/output card backup path
- In map settings for trace file path, audit file path, or work file path
- In the parameters of the environment functions (RUN, PUT, GET, EXIT, DBLOOKUP, and DBQUERY) within the map rules

With the Resource Registry, you define the various resource values once. Those values are resolved at run time (map execution) by the running engine (Command Server, Launcher, Transformation Extender Programming Interface (TXPI), WebSphere Message Broker), thereby eliminating the need to modify maps during design time.

With the Resource Registry, you can move your maps and systems to another environment, without the necessity of modifying your source files. Also, whenever you must change a user's password for a database, the Resource Registry helps you save a lot of time and problems.

After you define the resource variables, you must define virtual servers, one for each deployment environment. Each resource alias has a specified value for each virtual server.

If you want to protect sensitive data, such as passwords, you can encrypt resource variable values. Those values are only visible in the Resource Registry interface and encrypted in the resource variable files. Also, when you select the Encrypt option, the value of this resource is masked anywhere that it is displayed in audit logs and trace files.

Using resource names in Design Studio

To use a defined resource name, go to the place where you want to use it and put the alias enclosed with the percentage (%) symbol. The following figures show several examples of using resource aliases. You can concatenate a resource

alias with plain text and with other resource aliases, such as `C:\%ProjectName%\OutputDirectory\%FileName%.txt`.

In the Input and Output card settings, you can use resource names to set adapter commands similar to the examples in Figure 5-169 and Figure 5-170 or to specify a backup location. In Figure 5-169, FilePath is defined by using the OutPath resource variable. Also the value for Directory is specified as the BackupPath resource variable.

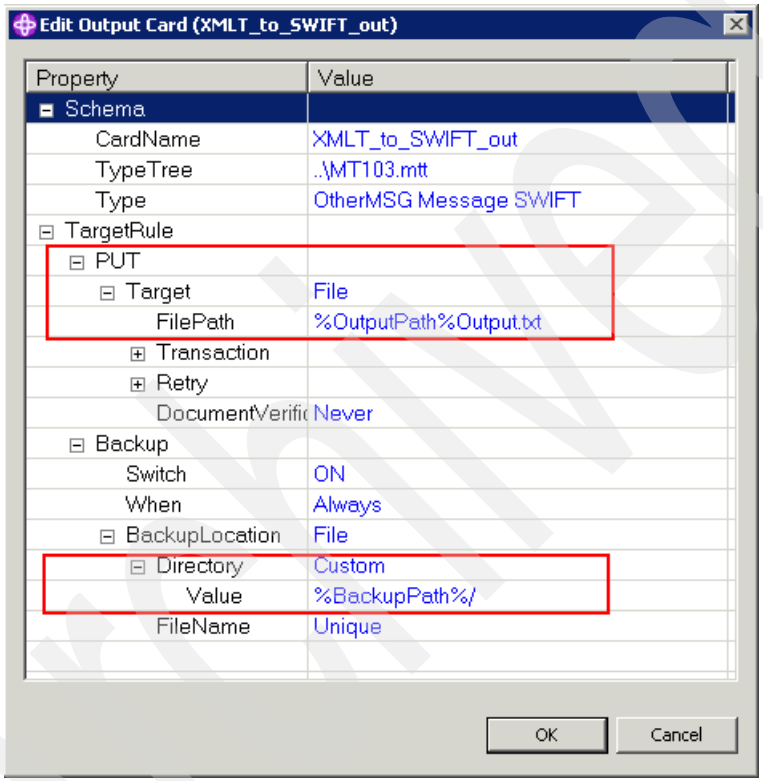

Figure 5-169 Using the resource variable in the card settings

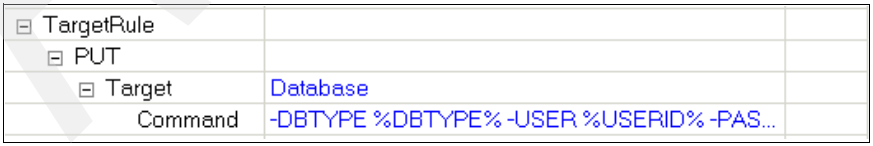

Figure 5-170 Using resource variables for setting the database adapter

For map settings, you can use the aliases to specify the audit location (as shown in Figure 5-171), trace location, and workspace files.

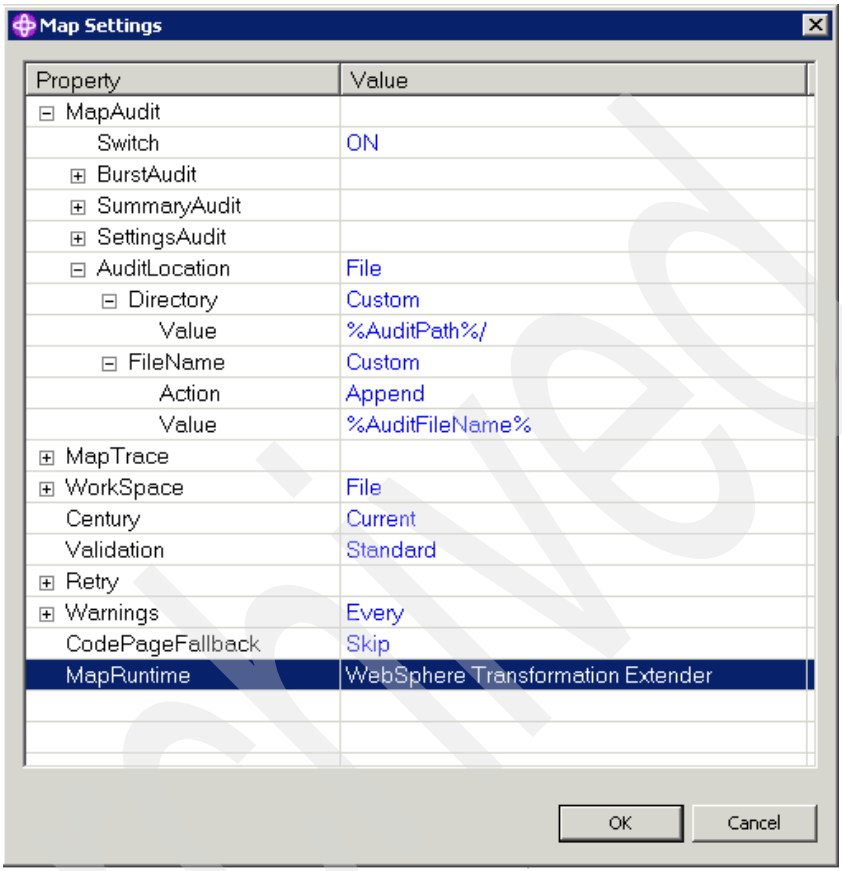

Figure 5-171 Resource aliases in Map Settings

The resource aliases can be used in the environmental functions such as RUN, PUT, GET, DBLOOKUP, DBQUERY, DDEQUERY, and EXIT. In the example in Figure 5-172, the text `IBM Redbook` is saved in the file that is named as defined in the resource variable FileName.

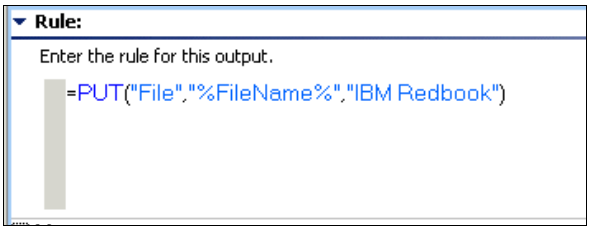

Figure 5-172 Using resource aliases in mapping rules

Note: Another good usage of the resource aliases is the possibility of switching on or off the trace for adapters. Because of performance reasons, do not use this method in the production environment to switch the trace on to run all the time. However, sometimes in case of problems, turn on the trace for a short period of time. Instead of modifying the map source and replacing the compiled map, use the resource alias and change its value whenever you need.

Consider the following example, in which we use an alias TRACE as a placeholder. The value can be empty, except when you need to trace the connection with the Queue Manager.

```
PUT ("MQS", "-QMN myqueuemgr -QN chips_queue %TRACE%", PACKAGE
(PaymentMessage:CHIPS_Payment_Message))
```

Sometimes the resource aliases can be unresolved. It can happened in two cases:

- An alias is not defined in the Resource Name file. For example, in the command, if you used `C:\%FileName%.txt`, it changes to `C:\.txt`.
- Aliases are not used, which can indicate the following meanings:
  - No configuration file was specified.
  - The configuration file does not exist.
  - The configuration file or resource file is invalid.

  In these situations, your command remains unchanged with percentage (%) characters. Therefore, in the same example, WebSphere Transformation Extender creates the `%FileName%.txt` file.

Resource Registry files

The Resource Registry creates the following files:

- * .mrn (resource name) file

  Resource name files contain a named set of virtual servers and a named set of resources.

- * .mrc (resource configuration) file

  Resource configuration files specify the associated resource name file and which virtual servers are active for a map server.

Both resource name and configuration files are in the XML format.

Using the Resource Registry interface

Resource Registry is a standalone application for editing Resource name and Resource configuration files. The installation program adds the entry for the Resource Registry to the WebSphere Transformation Extender programs menu (Figure 5-173).

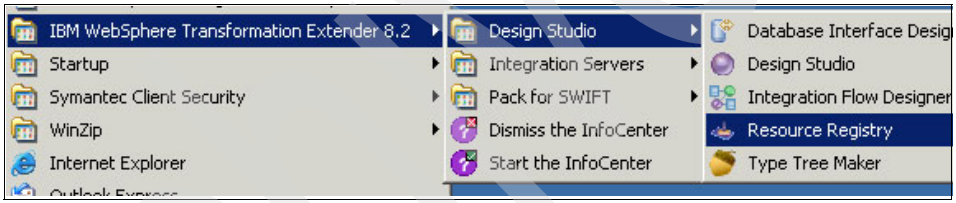

Figure 5-173 The Resource Registry in Program menu

The Resource Registry is delivered with the WebSphere Transformation Extender Design Studio, WebSphere Transformation Extender for Integration Servers, and WebSphere Transformation Extender for Application Programming.

For UNIX platforms, you can access the Resource Registry from `install_dir\resourceregistry.sh`.

After opening Resource Registry, you see a tree-structured navigator window (Figure 5-174) where you can work with resource files and configuration files.

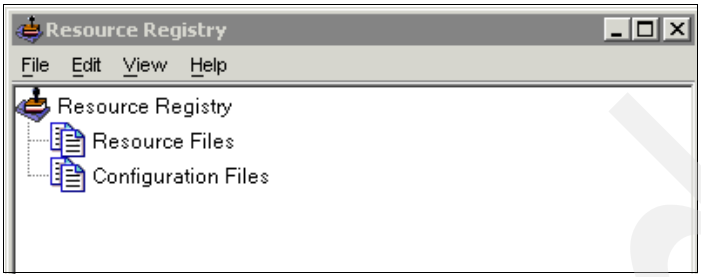

Figure 5-174 Resource Registry interface

Resource name files

The resource name file contains the following information:

- Resource names
- Resource values
- Virtual servers

To create a resource name file:

1. In the navigator, select **Resource Files**.
2. Select **File** → **New**.
3. In the Save as window:
   a. Specify the location of the resource name file by browsing your file structure
   b. In the File name field, enter the name of the resource name file.
   c. Click **Save as**.

The resource name file is displayed in the navigator, and you can start to define new resource names, virtual servers, and resource values.

To create a resource name:

1. In the navigator, right-click **Resources** and select **Add**.
2. In the Add New Resource window (Figure 5-175), in the Name field, enter the name of the resource and then click **OK**.

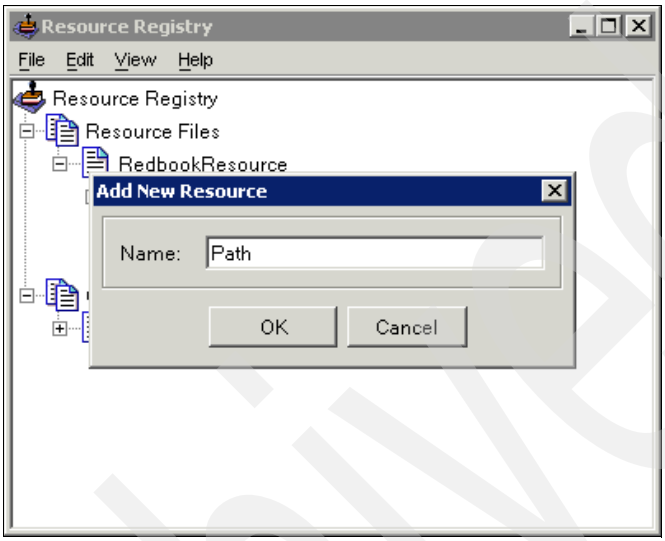

Figure 5-175 Add New Resource

The resource is displayed in the navigator (Figure 5-176).

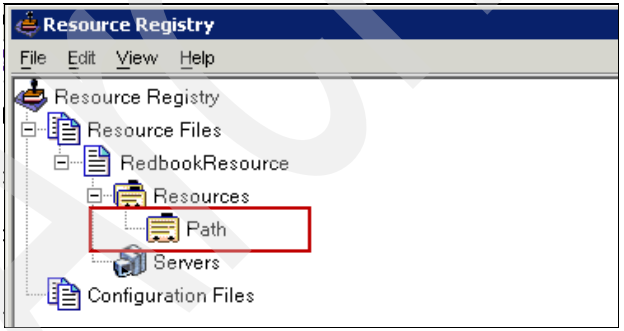

Figure 5-176 The new resource name

Virtual servers are created to represent the deployment environment. To create a server resource:

1. In the navigator, select **Servers**.
2. Select **Edit** → **Add**.
3. In the Add New Server window (Figure 5-177), In the Name field, type the name of the server resource and then click **OK**.

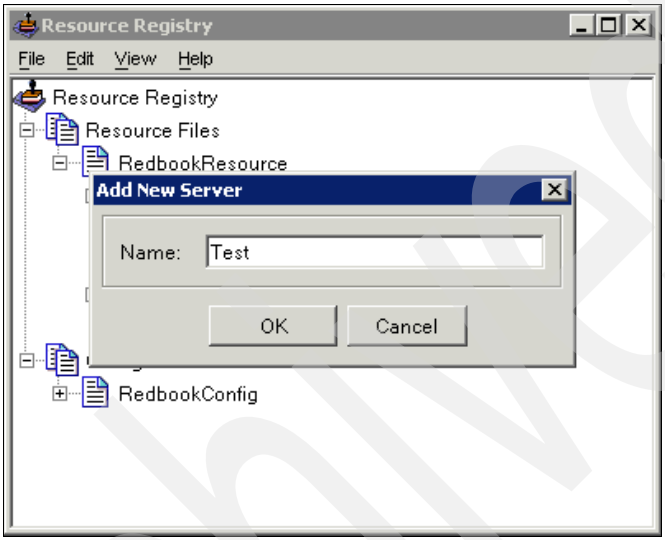

Figure 5-177 Add New Server window

The server resource is displayed in the navigator (Figure 5-178).

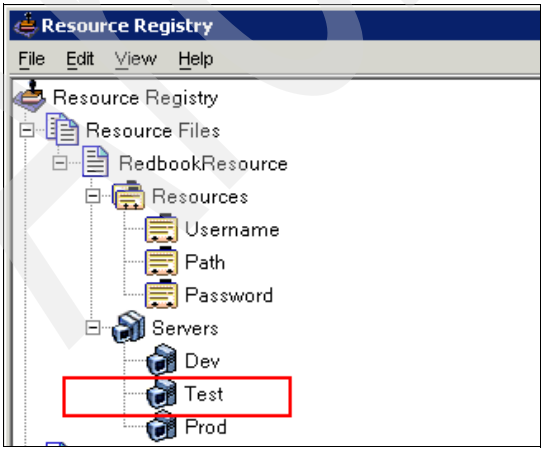

Figure 5-178 New Virtual Server in the navigator

Defining resource values

After you define the resource and virtual server names, the value for each resource name and virtual server combination must also be defined. The resource value can be encoded in the .mrn file as a security precaution.

To define the resource value:

1. In the navigator, select the resource name.
2. Select **Edit** → **Edit**.
3. in the Edit Resource window:
   a. In the Resource Value field, enter the value of the resource for a specified server.
   b. In the Encryption field, select **ON** to protect the resource name file or accept **OFF** as the default.
   c. Click **OK**.

You can open the Edit Resource window from two perspectives:

- From the Resource perspective, where you can edit values of the same Resource for different servers as shown in Figure 5-179

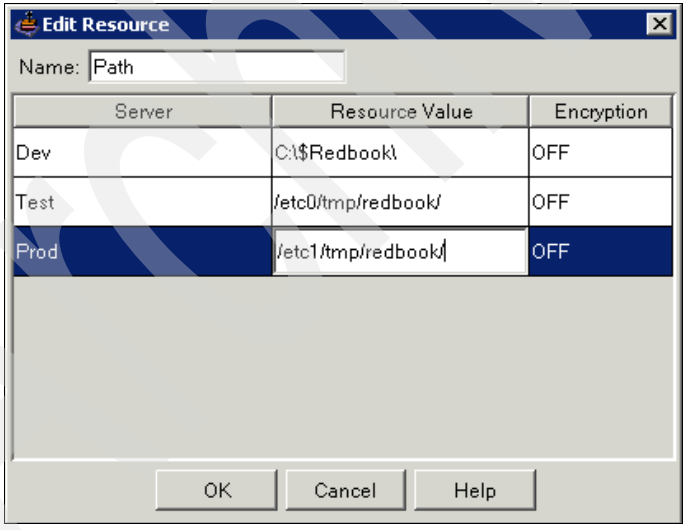

Figure 5-179 Edit Resource window

- From the Server perspective, where you can edit the values of different resource names for one virtual server, as shown in Figure 5-180

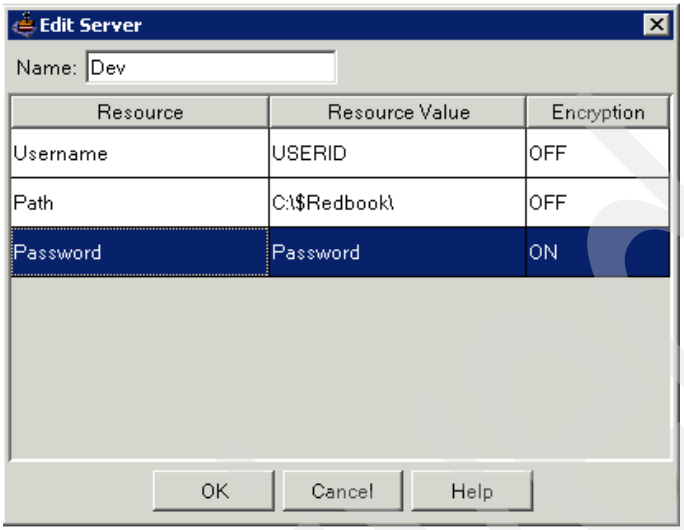

Figure 5-180 Editing server values for different resources

Resource configuration file

After you define your resources and their associated virtual servers, you must create a configuration file. This file is used by the Command Server, Launcher, Transformation Extender Programming Interface (TXPI), Java API, and Map Editor to determine which virtual servers are active for that engine when executing maps.

The resource configuration file is the key to tying everything together. This file defines the active virtual server, in a resource name file, for the map server and the available resources for that virtual server.

The .mrc file contains the following specifications, among others, for the map server:

- Active virtual servers for a map server
- Associated .mrn file

Resource name files are associated with the Launcher at the system level.

The specified active resource configuration file is used at run time for dynamic resource resolution. Because a .mrn file and its file path are specified, the .mrc file is platform dependent.

To create a resource configuration file:

1. In the navigator, select **Configuration Files**.
2. Select **File** → **New**.
3. In the Save As window:
   a. Specify the location of the configuration name file by browsing your file structure.
   b. In the File name field, enter the name of the configuration file.

      Note: A default resource configuration file can be specified for Command server and for Launcher in the `dtx.ini` file. uncomment the setting. Then specify a valid path and .mrc file name at both levels if necessary (lines 06 and 51 of the `dtx.ini` file).

   c. Click **Save As**.

The resource configuration file is now displayed in the navigator (Figure 5-181).

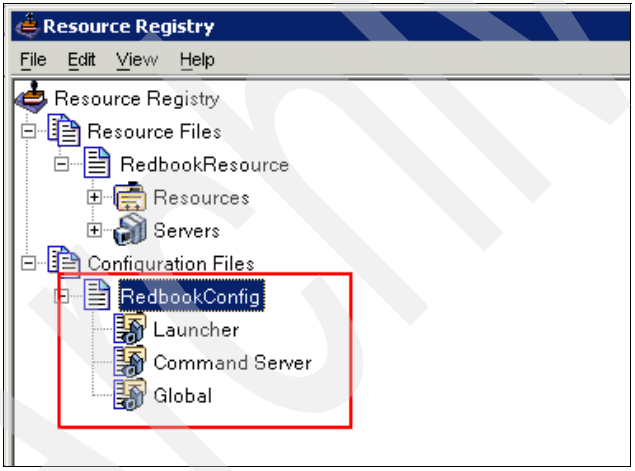

Figure 5-181 Resource Registry configuration files

The resource registry configuration file now contains three sections, each of which is used in different execution environments:

- Launcher

  In this section, you can specify the Resource Registry name file (.mrn) and active virtual server for maps that are run by Launcher. If you run several systems on the same Launcher, the settings can be provided for each system independently.

The .mrn file: At run time, if a .mrn file is not specified for Launcher, the .mrn file specified for Global is used by default. If .mrn files are specified for both Launcher and Global, the .mrn file specified for Launcher takes precedence.

- Command Server

  In this section, you can specify the Resource Registry name file (.mrn) and active virtual server for maps executed by the Command Server.

- Global

  This section is used for the maps that are run by the Design Studio, Transformation Extender Programming Interface (TXPI), Java API, and WebSphere Message Broker.

To add a system for the Launcher:

1. In the navigator, select **Launcher**.
2. Select **Edit** → **Add**.
3. In the Add New System window, click **Browse** to select the system file (.msl) to associate with the Launcher. Then click **OK**.

The associated system is displayed in the navigator (Figure 5-182).

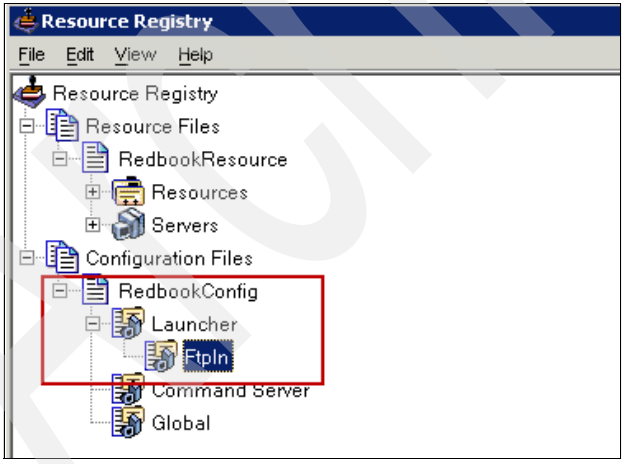

Figure 5-182 System in Launcher

To activate a server for the Launcher:

1. In the navigator, select the system.
2. Select **Edit** → **Edit**.
3. In the Edit Resources window, click **Add**.
4. In the Select window, select the resource name file (.mrn) to associate with the map server.
5. In the Edit Resources window, in the Active Virtual Server field, select the server to activate for the map server and then click **OK**.

The specified .mrn file and active virtual server are associated with the system.

A resource name file is used by the Command Server to determine which virtual servers are active when executing maps.

At run time, if the .mrn file is not specified for Command Server, the file specified for Global is used. If .mrn files are specified for both Command Server and Global, the Command Server file takes precedence.

To add a resource name file (.mrn) and activate a virtual server for the Command Server and Global:

1. In the navigator, select **Command Server** or **Global**.
2. Select **Edit** → **Edit**.
3. In the Edit Resources window, click **Add**.
4. In the Select window, select the resource name file (.mrn) to associate with the Command Server. Click **Select**.

5. In the Edit Resources window, in the Actual Virtual Server field, select the server to activate for the Command Server (Figure 5-183) and click **OK**.

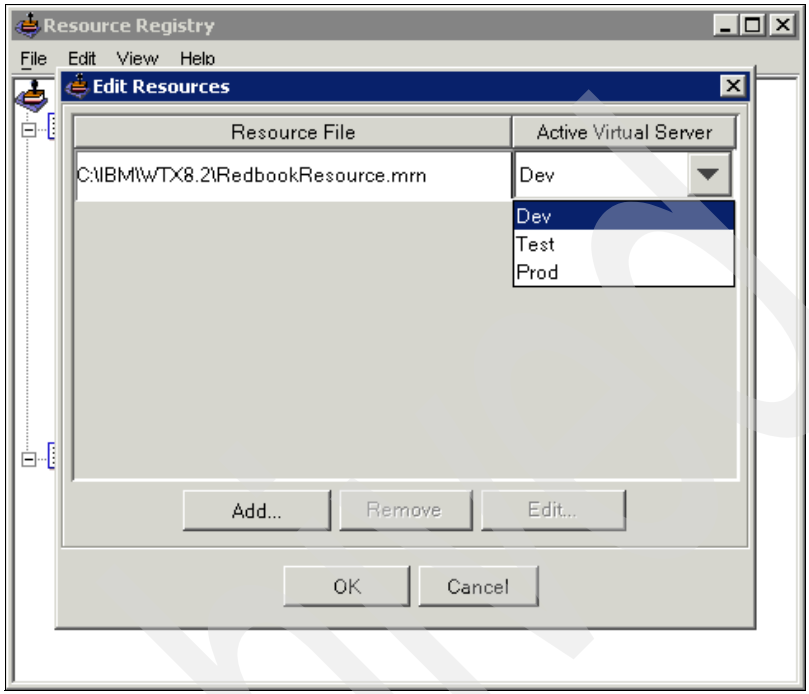

Figure 5-183 Specifying the active virtual server

The specified .mrn file and active virtual server are associated with the Command Server.

Specifying configuration files for the execution environment

After you configure and save the Resource Registry files, you must configure the execution engine to use the correct resource configuration file (.mrc). In this section, we explain where you can specify the .mrc file, depending on which method of execution you use.

Design Studio

A .mrc file can be specified for maps as a run option. When a map runs, the .mrc file defers to the .mrn file, and the virtual server that is used is derived from the Global setting.

To access run options in the Design Studio, from **Window** → **Preferences** and then navigate to **Transformation Extender** → **Map** → **Run Options** → **Configuration File** (Figure 5-184).

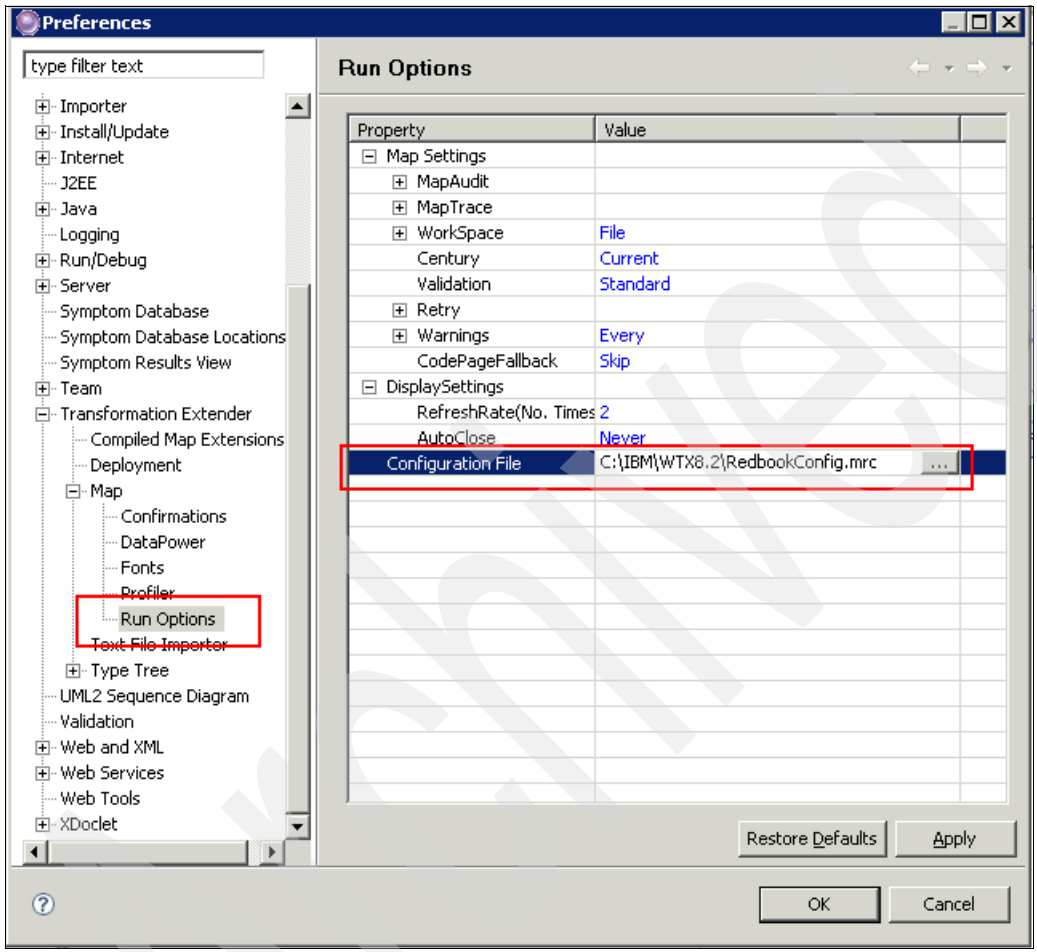

Figure 5-184 Setting the resource configuration file in the WTX Design Studio

Launcher Administration

The associated .mrc file for the Launcher is specified on the General tab in the Launcher Administration interface. See the Launcher documentation for more information. In case of executing maps in Launcher, the .mrn file and active virtual server are specified for each *.msl system file.

Initialization file

If a .mrc file has not been specified for the Command Server or Launcher through an application option, then the map server looks in the `dtx.ini` file for the specified .mrc file. A `resource.mrc` default file name is specified in the `dtx.ini` file. This .ini file is installed in the WebSphere Transformation Extender installation directory. If you create an .mrc file by using a different name, and do not specify the file name through an application option, then you must manually modify the .ini file as shown in Example 5-7 and Example 5-8.

Example 5-7 The dtx.ini file with the configuration file specified for Command Server

```
[Execution]
;specify the locale of the product (possible values de,en,es,fr,it,ja,ko,pt_BR,zh_CN
,zh_TW)
;locale=en

; this defines what resource configuration file to use for the Command Server:
ResourceCfgFile=C:\$Redbook\RedbookConfig.mrc
...
```

Example 5-8 The dtx.ini file with the configuration file specified for Launcher

```
[Launcher]

...

; this defines what resource configuration file to use for the Launcher:
ResourceCfgFile=C:\$Redbook\RedbookConfig.mrc
...
```

In the `dtx.ini` file, the .mrc file is specified in the *Execution* section for the Command Server and in the *Launcher* section for the Launcher. To determine where to specify the .mrc file, search for the ResourceCfgFile string and modify the current default .mrc file name.

Transformation Extender Programming Interface

If you want to specify an *.mrc file for maps that are executed from the Transformation Extender Programming Interface, you must provide the configuration file during initialization of the WebSphere Transformation Extender environment (Example 5-9 on page 349). The .mrn file and the active virtual server are derived from the Global setting of the .mrc file.

Example 5-9 Initialization of the WebSphere Transformation Extender environment with Resource Registry configuration

```
public static void main(String[] args)
{
String config_file = "C:\\$Redbook\\RedbookConfig.mrc";
try
{
// Initialize the API
MMap.initializeAPI(config_file);
...
```

WebSphere Message Broker

In case of the execution maps in WebSphere Message Broker, you can specify the .mrc file for an execution group. See 4.4.2, “Using WebSphere Transformation Extender for Integration Servers with WebSphere Message Broker” on page 135, for more information about how to set the necessary properties. The .mrn file and the active virtual server are derived from the Global setting of the .mrc file.

5.4 Command line utilities

WebSphere Transformation Extender provides a set of command line utilities that perform various functions as in the following examples:

- Analyzing type trees
- Importing to and exporting type trees
- Converting type trees
- Converting type tree properties from bytes to characters
- Compiling maps
- Importing to and exporting map source files
- Deploying systems
- Importing to and exporting systems
- Calculating suggested memory page size and count for maps
- Analyzing map execution behavior
- Converting XML type definitions
- Creating a map that can transform any input data into XML output
- Modifying resource values in .mrn and .mrc files

Some of the commands offer basic functions that are performed by the Design Studio, but that can be executed from the command line or within a command script without running the Design Studio GUI applications. Other commands offer other related functions and are executed in the same way.

Utility commands are available for the Type Tree Editor, Map Editor, and Integration Flow Designer applications, as well as for other related tasks.

After executing a utility command, a value is returned to the ERRORLEVEL environment variable indicating the success or failure of the execution. The ERRORLEVEL variable is a Windows environment variable that contains the return code of the last DOS command that you executed.

To retrieve the value of the ERRORLEVEL variable, enter the following DOS command:

```
ECHO %ERRORLEVEL%
```

Utility commands return the valid values (shown in Table 5-2) in the ERRORLEVEL variable.

Table 5-2 ERRORLEVEL variable return codes

Return code	Description
0	This value indicates that the utility command was successful.
1	This value indicates that the utility command was not successful

The ERRORLEVEL variable can also be used in batch files to retrieve the value of the return code.

Table 5-3 briefly defines the commands. See the WebSphere Transformation Extender documentation for details about any of the utility commands.

Table 5-3 Utility commands

Application or utility command	Usage
Type Editor	
tanalyze	Analyzes type trees
timport	Imports to type trees
texport	Exports type trees
dsxmlconv	Converts type trees that were generated in older versions (before V8.0)
tbcconv	Converts type tree properties from bytes to characters

Application or utility command	Usage
Map Editor	
mcompile	Compiles maps
mimport	Imports to maps
mexport	Exports maps
Integration Flow Designer	
sdeploy	Deploys systems
msdimport	Imports to system files
msdexport	Exports system files
Map Tuning	
dtxpage	Calculates suggested settings for memory page size and page count for maps
dtxprog	Profiles and analyzes map execution behavior
Utility commands for XML	
dsmapconv	Automatically converts new XML type definitions in existing maps
dtxany2xml	Automatically produces a map that can transform any input data into XML output
Resource Registry utility commands	
ResourceRegistry Helper.bat ResourceRegistry Helper.sh	Enables you to modify a resource value in a resource name file (.mrn) or resource configuration (.mrc) file

6

WebSphere Transformation Extender on System z

The WebSphere Transformation Extender engine is available on the IBM System z platform for embedding in COBOL, C, or PL/I applications. It is also available on the System z platform for batch processing and integrating with IBM DB2 software, and for participating in IBM CICS or IBM IMS transactions.

In this chapter, we explain WebSphere Transformation Extender Version 8.2 support on the System z platform and on Linux on System z. The topics covered include:

- Installation on z/OS
- Transformation and execution on z/OS
- Running a map in batch mode
- Input data validation and reading and writing XML data
- Using the Platform APIs on z/OS
- The CICS execution option
- IMS TM and IMS/DC
- The UNIX System Services Command Server
- Using WebSphere Transformation Extender on Linux on System z

WebSphere Transformation Extender Version 8.2 also supports Linux on System z technology. In this chapter, we also discuss the specific topics linked to z/OS.

6.1 Installation on z/OS

The installation process uses the standard SMP/E for z/OS rules. The result of the installation process is a set of partitioned data sets (PDS) as shown in Figure 6-1. The high qualifier name might be different.

```
DTX.REL8203.SDTXCLST
DTX.REL8203.SDTXCNFG
DTX.REL8203.SDTXHFS
DTX.REL8203.SDTXHFS.DATA
DTX.REL8203.SDTXLOAD
DTX.REL8203.SDTXLOD2
DTX.REL8203.SDTXMAPS
DTX.REL8203.SDTXPENU
DTX.REL8203.SDTXSAMP
```

Figure 6-1 z/OS installed PDS

Note: The references to documentation in this chapter are to the appropriate documentation PDF file from the WebSphere Transformation Extender Online Library.

The installation output has the following contents:

- SDTXCLST and SDTXPENU are for the WebSphere Transformation Extender Launcher administration console.
- SDTXHFS and SDTXHFS.DATA are for the UNIX System Services version.
- SDTXLOAD is the WebSphere Transformation Extender library where Command Server and WebSphere Transformation Extender utilities are located.
- SDTXLOD2 is the WebSphere Transformation Extender library for CICS, compiled without optimization XPLINK(ON) parameter.
- XPLINK(ON) parameter.

  Note: The DTX.SDTXLOAD load library contains files that support the use of XPLINK. For CICS, if you want to use XPLINK, you must add the DTX.SDTXLOAD load library to the DFHRPL concatenation.

- SDTXSAMP is the PDS where all examples, readme, data structure definitions, sample maps, and installation and post-installation job control

language (JCL) files located. See Appendix A, “Running on z/OS” on page 679, for a full list.

- SDTXMAPS contains all map source and type tree files of examples to be downloaded in the Design Studio for testing and training.

For each subsystem, a post-installation process must be done. JCL files are provided in SDTXSAMP:

- z/OS post-installation mainly consists of binding WebSphere Transformation Extender components to DB2. This binding is common to all environments and is required if you use DB2 in maps or adapters.
- CICS must be configured, and a VSAM file must be created, to receive map files.
- IMS must be configured, and a COBOL or C program must be adapted. This process is described in the *IMS/DC Execution Option* guide (`1103.pdf` file) at the following address:

  `ftp://ftp.software.ibm.com/software/websphere/integration/wdatastagetx/`
- For UNIX System Services, you must create a runtime environment with appropriate variables and directories.
- Optional: If DB2 and WebSphere MQ adapters or functions are used, the corresponding library path must be added to the LIBPATH environment variable.

All post-installation processes are described in detail in the *z/OS Configuration* guide (`1145.pdf` file) in the IBM WebSphere Transformation Extender Online Library.

6.1.1 Prerequisites for running WebSphere Transformation Extender on z/OS: XML Toolkit v1.9

Check if the mandatory IBM XML Toolkit for z/OS load library (default name is IXM.SIXMLOD1) has been added to the system link list or the TSO logon PROC STEPLIB. If not, you must add it to all JCL files and scripts that launch WebSphere Transformation Extender.

Figure 6-2 shows the error message that is issued if the XML Toolkit is not present.

```
04.49.35 JOB15564 $HASP165 WTXEX    ENDED AT WTSC59 - ABENDED S000
U4038 CN(INTERNAL)
 ***
```

Figure 6-2 Error message if XML Toolkit is absent

Figure 6-3 shows the message in the log in SYSOUT DTX.

```
****************************** TOP OF DATA***************************
CEE3501S The module IXMI33XC was not found.
         From entry point callEngine at statement 474 at compile
unit offset +00
       address 2130182C.
************************** BOTTOM OF DATA ***********************
```

Figure 6-3 Log message from SYSOUT DTX

Important: The XML Toolkit is mandatory, even if you do not use XML files.

6.2 Transformation and execution on z/OS

Now that we have discussed the key points of the installation process for WebSphere Transformation Extender on z/OS, we focus on how to execute the transformations. We begin by looking at how to administer WebSphere Transformation Extender by using z/OS administrative tools and review the adapters that are included for transformations between sources and targets.

6.2.1 z/OS administration

Administration of WebSphere Transformation Extender on z/OS is done by using standard tools for administering the z/OS platform, including ISPF, SDSF panels, and so on.

Command Server is a batch task and has no permanent active processes as shown in Figure 6-4 on page 357. WebSphere Transformation Extender is only active when called by the JCL. When map transformation ends, WebSphere Transformation Extender is no longer active.

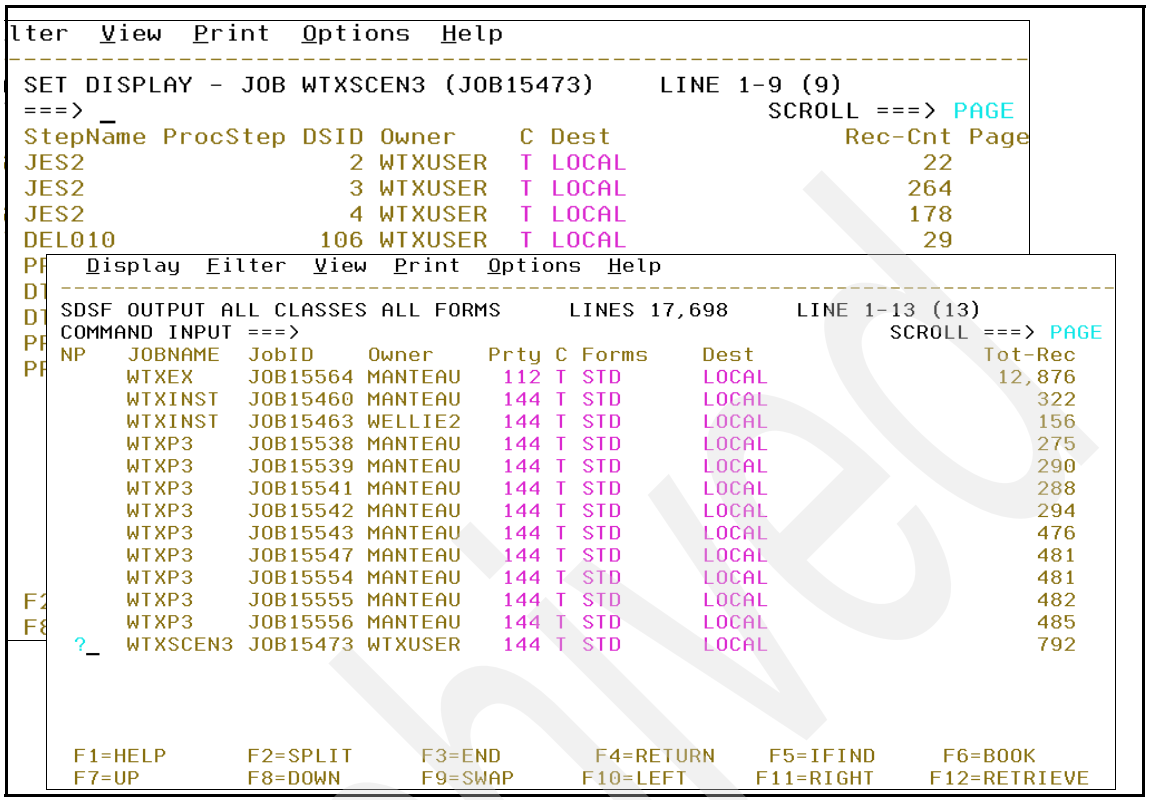

Figure 6-4 Jobs using WebSphere Transformation Extender

The WebSphere Transformation Extender Launcher process is always active and waiting for events to occur. There are three modes to handle the process:

- Manually with a command line (start, stop, pause)
- Through an ISPF panel
- Through a deported X Window System

Note: Before you use the z/OS native administration interface for the IBM WebSphere Transformation Extender with Launcher, make sure that the IBM XML Toolkit for z/OS load library (default name is IXM.SIXMLOD1) is added to the system link list or the TSO logon PROC STEPLIB.

6.2.2 z/OS adapters

All available adapters are delivered with each version of WebSphere Transformation Extender.

z/OS batch WebSphere Transformation Extender Command Server supports the following sources and targets natively:

- Files (QSAM, VSAM, UNIX System Services)
- DB2 (native or Open Database Connectivity (ODBC))
- FTP
- WebSphere MQ
- Sink

UNIX System Services Command Server and Launcher supports the following sources and targets:

- Files (hierarchical and native file systems)
- DB2 (native or ODBC)
- Oracle
- FTP
- WebSphere MQ
- CICS
- EMAIL
- Sink

6.3 Running a map in batch mode

The batch execution mode is invoked through JCL. WebSphere Transformation Extender can be called by Command Server or by a program written in COBOL, C, or PL/I by using the APIs. By using the EXIT function, existing COBOL, PL/I, and C programs can be called from a WebSphere Transformation Extender map.

Running a transformation on z/OS entails the following tasks:

1. “Building a map in Design Studio for the z/OS platform” on page 360
2. “Transferring a compiled map to a z/OS system” on page 362
3. “Transferring data files to a z/OS system” on page 363
4. “Modifying or creating a JCL file to execute as a batch job” on page 367
5. “Submitting a job” on page 380
6. “Examining the results” on page 380

To illustrate these different steps, the *reverse* sample map is used. This map is available on the system in the compiled form and as a map source with the needed type tree and data files (Figure 6-5).

```
From DTX.SDTXSAMP
DTXBMMVS - the compiled (DTXBMMMS) map  to be used directly
DTXRTEXT - the sample input data file

 From DTX.SDTXMAPS,to download to a Windows system
 DTXBMMMS   reverse.mms  - the map source for the REVERSE map
 DTXBMMTT   reverse.mtt  - the type tree used by the REVERSE map
Import all these files in a workspace of Design Studio
   . create a project
   . import files using "File System" option
Build and unit test the map.
```

Figure 6-5 Overview of process to use the sample

The reverse map reads lines that are composed of three words and separated by commas. The map reverses the order of the words for each line and writes the result in an output file. Two EXIT functions add text at the beginning and end of each line.

6.3.1 Building a map in Design Studio for the z/OS platform

Build and work with the map and then make it specific to run on the z/OS platform. Figure 6-6 illustrates the reverse map, which has been designed and tested in Design Studio.

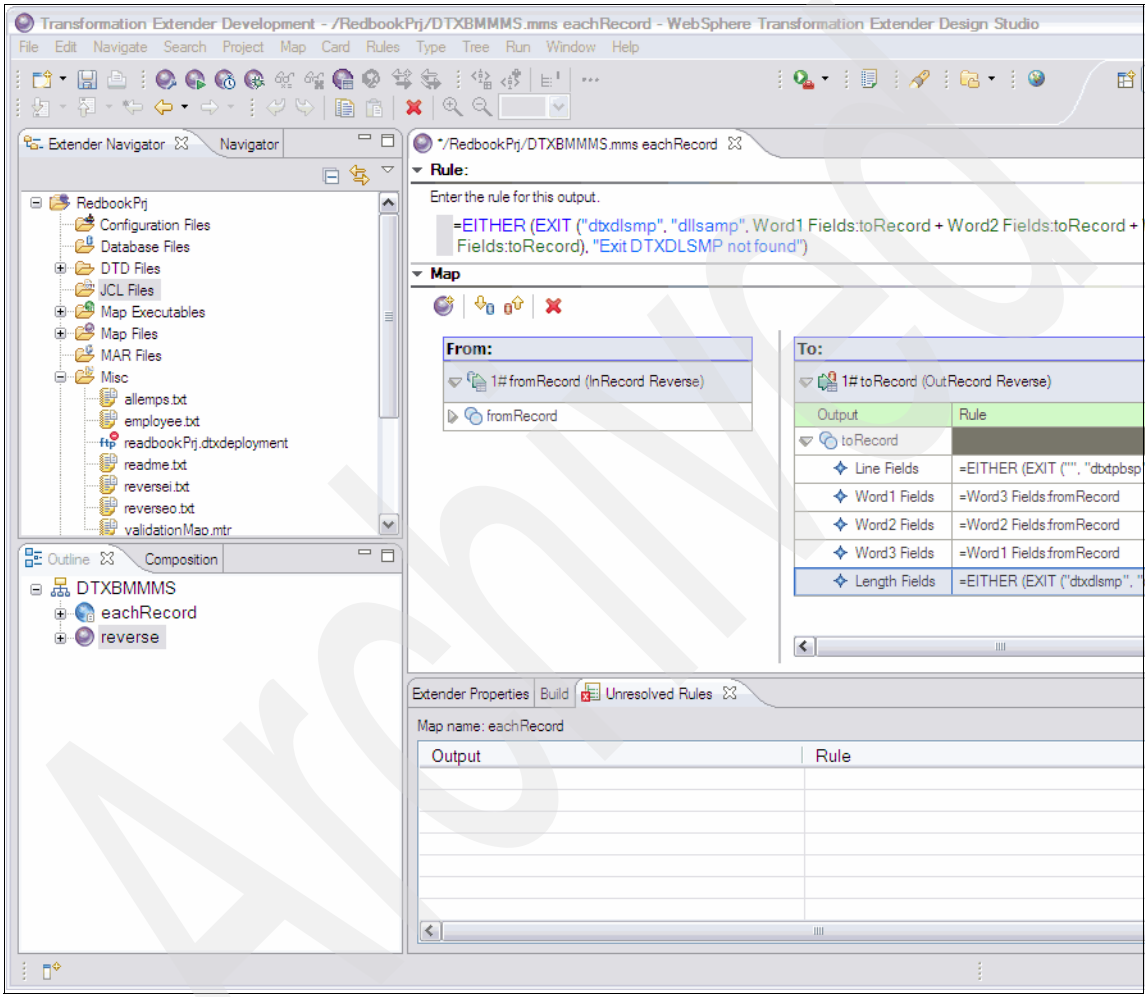

Figure 6-6 The sample reverse map in Design Studio

We assume that the map has been tested in Design Studio. Build the map to run it specifically on a z/OS system. Select the executable map **reverse**, right-click, and select **Build for Specific Platform** → **IBM z/OS** (Figure 6-7).

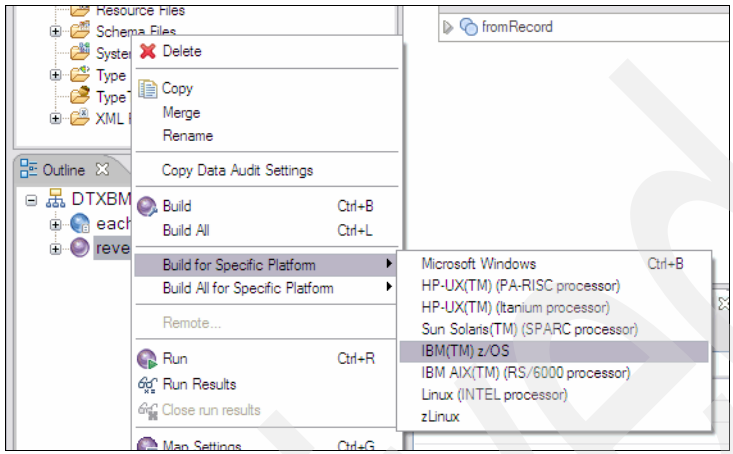

Figure 6-7 Building the map specifically for z/OS

The executable map for z/OS is generated with the .mvs extension and a JCL template. The .mvs map can be used for all z/OS systems including Batch-MVS, CICS, IMS, and UNIX System Services.

The .mvs map file: The .mvs map file is the only file to be deployed on the System z environment for a transformation. All industry packs (SWIFT, SEPA, and so on) are used only at development time. The *compiled* map file includes everything needed for execution.

There is an exception when using XML inputs. The schema or document type definition (DTD) file might be mandatory for validation or data definition.

By using the **`build`** command for z/OS map, the compiled map is placed in the Map Executable folder, and a JCL template is generated and placed in the JCL Files folder in Design Studio. This JCL template must be modified to be adapted to the map and files that are used. In this book, the sample JCL for the reverse map that is copied from the DSTXSAMP PDS is used.

6.3.2 Transferring a compiled map to a z/OS system

Transfer the `reverse.mvs` compiled map file in binary mode from the Windows environment to the z/OS environment as a sequential data set or member of a PDS. The setup must be a record format of fixed block (FB) and have a logical record length of 80. In addition, keep in mind the following points:

- Specify BINARY (not ASCII to EBCDIC translation and no elimination of carriage returns and line feeds).
- The file transfer must not use the CRLF option, or map corruption will result. Instead use either of the following options:
  - Use the IBM Personal Communication Facility.
  - Use standard FTP, which is available from a Windows command line

Example 6-1 illustrates FTP commands to load the map file.

Example 6-1 FTP command to load the map file

```
ftp <your-host>
UserName
Password
cd 'WTX.MAPS'
binary
quote site LRECL=80 RECFM=FB
PUT  C:\WebSphere\workspaces\wtx\Redbook\RedbookPrj\reverse.mvs REVERSE
quit
```

The quote site command: By using the FTP **`quote site`** command, you can set the file type and allocation on the host.

In the environment of the team who wrote this book, a PDS named WTX.MAPS was previously created to receive maps, as shown in Figure 6-8.

```
C:\Redbook>ftp  wtsc59oe.itso.ibm.com
Connected to wtsc59oe.itso.ibm.com.
220-FTP Server (user 'amanteau@fr.ibm.com')
220
User (wtsc59oe.itso.ibm.com:(none)): wtxuser
331-Password:
331
Password:
230-220-FTPD1 IBM FTP CS V1R9 at wtsc59.itso.ibm.com, 19:17:11 on 2008-09-23.
230-WTXUSER is logged on.  Working directory is "/u/wtxuser".
230
ftp> cd 'WTX.MAPS'
250 The working directory "WTX.MAPS" is a partitioned data set
ftp> binary
200 Representation type is Image
ftp> quote site LRECL=80 RECFM=FB
200 SITE command was accepted
ftp> PUT  C:\Websphere\workspaces\wtx\Redbook\RedbookPrj\reverse.mvs REVERSE
200 Port request OK.
125 Storing data set WTX.MAPS(REVERSE)
250 Transfer completed successfully.
ftp: 2577 bytes sent in 0.00Seconds 2577000.00Kbytes/sec.
ftp> quit
221 Quit command received. Goodbye.

C:\Redbook>_
```

Figure 6-8 Uploading a map in a previously created PDS named WTX.MAPS

Important: By not using the correct format to upload, unpredictable output might result with a "`cannot read map`" message.

6.3.3 Transferring data files to a z/OS system

File names on z/OS are limited to eight characters. Usually data is stored in a native sequential data set, named with different levels:

- WTX.DATA.APP1.DATA1
- WTX.DATA.APP1.DATA2
- WTX.DATA.APP2.DATA1

The data set defines the physical names of data (DSN). Other possibilities are a member of a PDS, PDSE, VSAM, or a UNIX (UNIX System Services) reference.

Important: Take note of the format used for a data set. FB is used for fixed length records, and VB is used variable length records. This property has an impact on the end-of-record behavior of WebSphere Transformation Extender.

The end-of-record feature

Data files that are used by WebSphere Transformation Extender can originate from different systems:

- Input files for the map are generated by a z/OS application.

  These files do not have any end-of-record characters. Fixed length records are generally padded with spaces. Variable length records are padded with null characters (hex 40).

  For WebSphere Transformation Extender, check that the type tree reflects the exact length of the record. Do not use any record separator characters in the initiator or terminator properties.

- Input files for the map have Windows or UNIX sources.

  These files generally have an end-of-record character such as <CR><LF> (hex 0D, 0A) or <NL> (hex 15). The type tree that describes these files has either initiator or terminator values for each record property. Because z/OS files do not have these characters, input data validation fails when executing the map. Figure 6-9 shows an example of a typical error message.

```
.19.01 JOB15567 $HASP165 WTEX    ENDED AT WTSC59 MAXCC-1 CN (Internal)
*
```

Figure 6-9 Typical end-of-record error

Additionally, the DTXLOG trace file shows a message similar to "`does not match TERMINATOR '<CR>'`", as shown in Figure 6-10.

```
************************ TOP OF DATA *************************
*** IBM WebSphere Transformation Extender 8.2.0.3(58) Trace
Report for Map REVERSE
(Level 2: Offset 9, len 10, comp 2 of 3, £1, DI 00000002:)
Data at offset 9 ('aardwolves') was found to be of TYPE
  X'0006' (Word2 Fields Reverse).
(Level 2: Offset 20, len 64, comp 3 of 3, £1, DI 00000003:)
Data at offset 20 ('aba         ') was found to be of TYPE
  X'0007' (Word3 Fields Reverse).
(Level 1: Offset 84, len 0, comp 1 of 1, £1, DI 00000004:)
Data at offset 84 (',abaca,abaci    ...') does not match
TERMINATOR '<CR> '
 of TYPE X'0003' (InRecord Reverse).
(Level 1: Offset 0, len 0, comp 1 of 1, £1, DI 00000001:)
COMPONENT number 1 of TYPE X'0002' (InFile Reverse):
occurrence 1 is optional and does not exist.
  INPUT 1 exists (800 bytes) but has no content.
  End of Validation messages for INPUT CARD 1.
  End of Execution messages.
```

Figure 6-10 Does not match terminator message

When you look at the input file in hex mode, you see that each line ends with a hex 40 character, as Figure 6-11 shows.

```
----------------------------------
aardwolf,aardwolves,aba
8898A99868898A99A8A688844444444444444
11946636B1194663552B12100000000000000
 ----------------------------------
abac,abaca,abaci
888868888868888844444444444444444444
1213B12131B121390000000000000000000000
 ----------------------------------
```

Figure 6-11 Input file in hex mode

If the type tree contains record separators, z/OS WebSphere Transformation Extender uses the **/V** command line option to simulate these separators, followed by a hexadecimal representation of the separator characters and the DDNAME

of the file (Example 6-2). To be compliant with the type tree data definition, you must use this option.

Example 6-2 Command line option to simulate separators

```
//   PARM='REVERSEB /VXOD,XOA REVERSEI '
```

By using the command line option, you add the <CR><LF> character to each end of record for the `REVERSEI` file.

Important: To use the **/V** parameter, the file (input or output) *must* be a variable record length file. This **/V** mechanism does not work with fixed length files.

Transferring VB formatted data: When transferring data with the VB format, do not forget to add 4 bytes to the length of the record in the RECL parameter when dealing with data formatted with a COBOL copybook definition (fixed length).

Data encoding

Depending on the type tree that is used in the input cards of the map, a data file can be transferred in ASCII mode, which means a conversion to EBBCDIC of all characters, or in BINARY mode, which means no conversion. If data is in the XML format, the encoding is defined in the prolog of the file.

Example 6-3 shows the FTP command for transferring the `revesei.txt` file in ASCII mode, which is the default.

Example 6-3 Transferring the revesei.txt file

```
cd 'WTX.DATA'
quote site RECL=72  RECFM=VB
put c:\ex\reversei.txt REVERSEI
```

Example 6-4 shows the command to transfer a file of fixed length fields, with a record length of 143 bytes, but with a line feed at the end of each record. The size of the file must allocate 10 cylinders with 5 extends.

Example 6-4 Transferring a file with fixed length fields

```
quote site RECL=147 RECFM=VB CYLINDERS PRIMARY=10 SECONDARY=2
put c:\mywork\orders.txt 'WTX.DATA.ORDERS'
```

Quotation marks in the file name: The usage of single quotation marks (' ') around a file name in the FTP command signifies an absolute reference. That is the first part of the name is the “high level qualifier.” Not using the quotation marks means a position of the file relative to the current position.

6.3.4 Modifying or creating a JCL file to execute as a batch job

As a template, use the sample JCL that is provided in DTX.SDTXSAMP or the generated JCL from Design Studio. To launch the transformation, a JCL built for WebSphere Transformation Extender includes the following parts:

- Job specifications
- WebSphere Transformation Extender library location
- XML Toolkit library location
- Location of map
- Location of input and output files
- Location of schemas and document type definition (DTD) files
- Execution step, WebSphere Transformation Extender commands
  - Attached files and location of the system files
  - Temporary work files

Modifying the sample JCL

Make a copy of DTX.DSTWSAM(DTXMBJCL). In our environment, a PDS named WTX.JCL has been created to receive all JCLs. This sample JCL uses variables to store the map location, input and output files, and so on. Used as a template, this file provides an easier way to adapt the JCL to a new map and to enhance readability.

Figure 6-12 shows the template.

```
//DTXBMJCL JOB
//*****************************************************************
//*    Licensed Materials - Property of IBM                       *
//*                                                               *
```

Figure 6-12 Job specification for default job card

Figure 6-13 shows how we changed it for our environment. See your z/OS administrator for all of these parameters.

```
//WTXMAP       JOB (999,POK),'LO6R',CLASS=A,REGION=64M,
//             MSGCLASS=T,TIME=10,MSGLEVEL=(1,1),NOTIFY=&SYSUID
/
```

Figure 6-13 Specific job card parameters entered

In the following sections, we focus on defining the location of maps, files, and so on by filling variables with corresponding DSN names.

WebSphere Transformation Extender library location and XML Toolkit library location

DTXLIB is the variable name for the WebSphere Transformation Extender load library in this JCL, as shown in Figure 6-14.

```
// SET DTXLIB=DTX.REL8203.SDTXLOAD <== DTX LOADLIB
```

Figure 6-14 Variable name for WebSphere Transformation Extender load library

The XML Toolkit is mandatory, but can already be defined in the link list of the system (Figure 6-15). In such case, it is not necessary to be declared in the JCL.

```
//  SET XMLLIB=IXM.SIXMLOD1                    <== XML Toolkit
```

Figure 6-15 Variable name for the XML Toolkit

These two variables are used later in the JCL as STEPLIB when defining the WebSphere Transformation Extender execution step (Figure 6-16).

```
//DTX   EXEC PGM=DTXCMDSV,REGION=0M,
//   PARM='REVERSEB /VXOD,XOA REVERSEI /VXOD,XOA REVERSEO'
//STEPLIB  DD  DISP=SHR,DSN=&DTXLIB
//         DD  DISP=SHR,DSN=&XMLLIB
```

Figure 6-16 Variables used in STEPLIB

Location of the map to run

Maps can be stored in a sequential data set or as member of a PDS as shown in Figure 6-17.

```
//  SET MAPLIB=WTX.MT1032.EMP.MAP
//  SET MAPLIB=WTX.MT1032.MAP(EMP)
```

Figure 6-17 Setting the map locations

The reverse map has been uploaded in WTX.MAPS PDS, but the same map is already provided as a sample in the DSTXSAMP PDS (Figure 6-18).

```
//  SET MAPLIB=DTX.SDTXSAMP(DTXBMMVS)
```

Figure 6-18 Using the sample map

Keep this line to use the provided compiled reverse map, or change the line to use the map that was just uploaded (Figure 6-19).

```
//  SET MAPLIB=WTX.MAPS(REVERSE)
```

Figure 6-19 Using the uploaded map from Design Studio

Location of the input/output files

Check that the input file exists (Figure 6-20).

```
//  SET MAPINPT=DTX.SDTXSAMP(DTXRTEXT)
```

Figure 6-20 The input file

Change the MAPOUT variable (Figure 6-21). The output file is allocated in the execution step.

```
//  SET MAPOUT=DTX.OUT.REVERSEO <== OUTPUT FILE 1
```

Figure 6-21 Setting MAPOUT

Note: A UNIX (UNIX System Services) sample path can be used for the location of the data set:

```
//PACS     DD PATH='/u/wtxuser/data/sepa/SEPAmsg10.xml'
```

This path can be useful for large files. Running with a native data set has a 2 GB limit, and running with a hierarchical file system (HFS) has a 4 GB limit.

Location of the schemas and DTD files

The map does not have any XML input or output verification. When using input XML data and the Xerces parser, the DDNAME schema or DTD file must be defined.

Invoking the WebSphere Transformation Extender Command Server

WebSphere Transformation Extender Command Server is called by using DTXCMDS, which is member of DTX.REL8203.SDTXLOAD that is declared in the DTXLIB variable. DTXCMDS has a PARM parameter that contains all commands and overrides for translation See Figure 6-22 on page 371.

DDNAMEs: All files must have a DDNAMEs reference. The DDNAME map is the first and mandatory parameter of the command line.

```
//*
//DTX   EXEC PGM=DTXCMDSV,REGION=0M,
//   PARM='REVERSEB /VXOD,XOA REVERSEI /VXOD,XOA REVERSEO'
//STEPLIB  DD  DISP=SHR,DSN=&DTXLIB
//         DD  DISP=SHR,DSN=&XMLLIB
//*
//* The map ddname specified in the JCL must match the map name
specified
//* on the command line defined in the PARM statement
//*
//*  A map can be coded in JCL in any of the
//* following three ways:
//*
//*    //REVERSE  DD DSN=DTX.SDTXSAMP,DISP=SHR
//*    The command line identifies the map by its ddname:
//*    e.g. PARM='REVERSE <option> <option> etc.'
//*
//*    //REVERSE  DD DSN=DTX.SDTXSAMP(DTXBMMVS),DISP=SHR
//*    The DD statement identifies the map as a member of
//*    a PDS and the command line identifies the map by its ddname:
//*    e.g. PARM='REVERSE <option> <option> etc.'
//*
//*    //MAPLIB   DD DSN=DTX.SDTXSAMP,DISP=SHR
//*    The DD statement identifies the PDS only and the member
//*    name is identified on the command line in parentheses
//*    following the ddname that identifies the PDS:
//*    e.g. PARM='MAPLIB(DTXBMMVS)'
//*
//REVERSEB DD  DISP=SHR,DSN=&MAPLIB
//*
```

Figure 6-22 Execution command in PARM statement containing DDNAME of a map

The DDNAME map must refer in the step to a DSN (physical location of the file), which is defined in the MAPLIB variable. Figure 6-23 shows the map execution command.

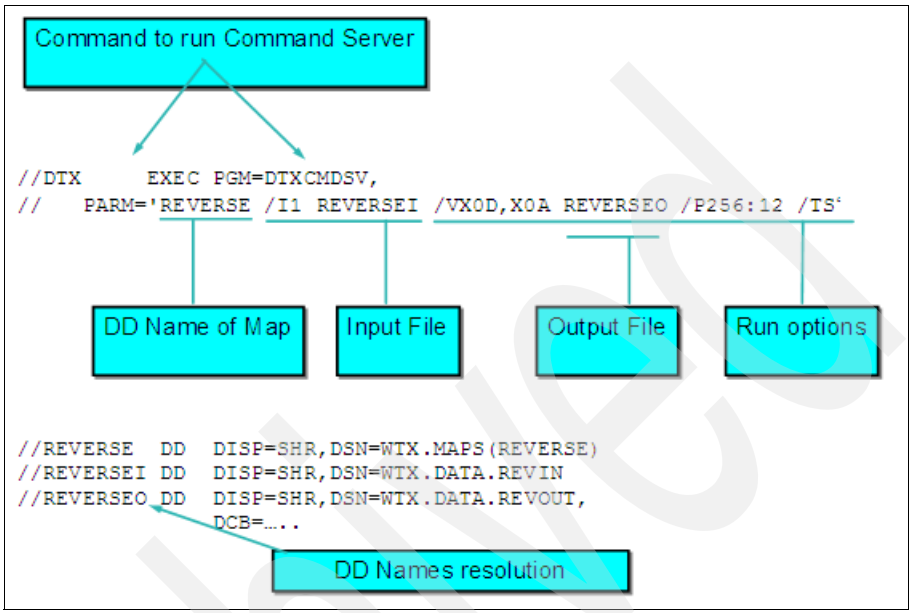

Figure 6-23 Map execution command

For an explanation of the parameters, see the *Command Server* guide (`1010.pdf` file) for specific z/OS options and the *Execution Command* guide (`1007.pdf` file) for more detailed explanation about common options at the following address:

`ftp://ftp.software.ibm.com/software/websphere/integration/wdatastagetx/`

Figure 6-24 on page 373 shows a summary.

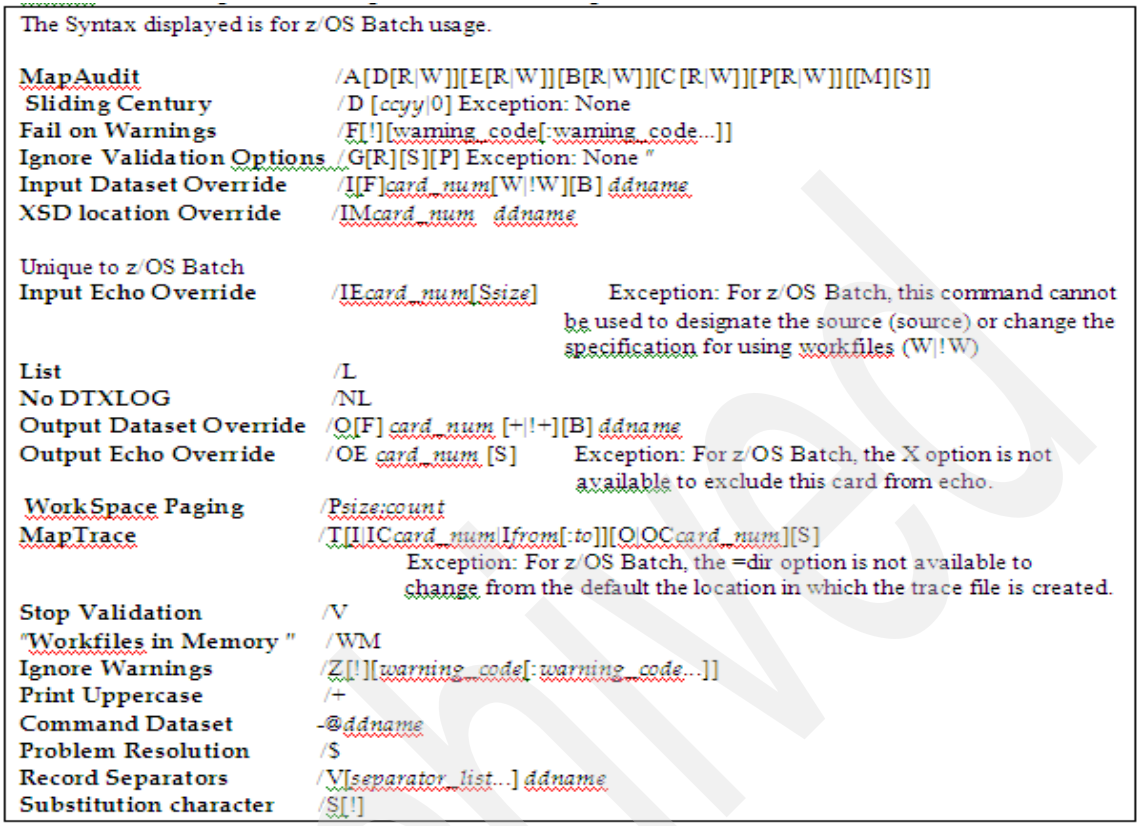

Figure 6-24 Override parameters specified in the map

All files in the PARM line, if needed, must refer to DDNAMEs. By default, the names for input and output DDNAMEs are deducted from the file name given at development time in the input/output card properties.

File name conversion: The source and target names are converted from Windows to z/OS:

- Drive and paths are trimmed.
- Extensions are dropped.
- All but the left eight characters of trimmed file name are ignored.
- Underscore (_) characters are converted to the number symbol (#).
- Lowercase is converted to uppercase.

Consider the example where the file name on Windows is `C:\mytestfiles\input\home_address.txt`. The DDNAME on z/OS is `HOME#ADD`.

The default DDName of sources and targets can be overridden at run time by using a specific option. To override with files, use the **/In** or **/On** option, where *n* is the card number.

To override with WebSphere MQ queues, use the `//IMMQSn` or `/OMMQSn` option, where *n* is the card number. See the documentation in the *Extender IBM WebSphere MQ Adapter* guide (`1059.pdf` file) at the following address:

`ftp://ftp.software.ibm.com/software/websphere/integration/wdatastagetx/`

DDNAMEs must have eight characters and must not be reserved by WebSphere Transformation Extender.

Reserved DD names:

- DTXDEBG is for aid in problem resolution.
- DTXAUD is for audit file.
- DTXLOG is for execution log.
- DTXTRCE is for trace output.
- DTXMRN is for DTX Resource Registry name file.
- SYSTMPnn, where nn is 01 - 99, is reserved for statically allocated data sets.
- SYSTMnnn, where nn is 100 - 999, is reserved for statically allocated data sets.

In the current example, DDNAMEs are not overridden. The default names are REVERSEI and REVERSEO, which are directly converted from file names that are used in the input and output cards. See Figure 6-25.

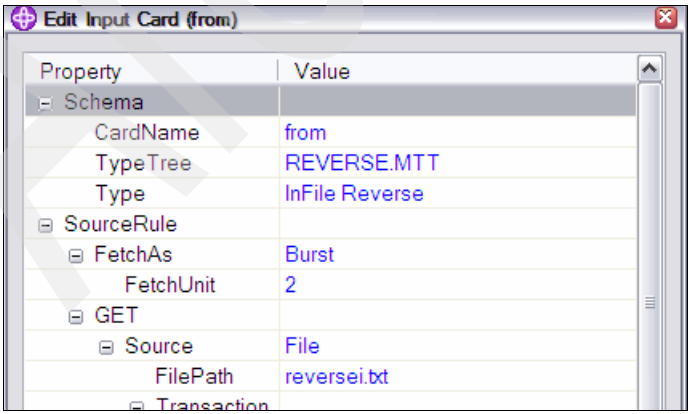

Figure 6-25 File name in the input card

The **/V** parameter is used for both input and output files to emulate the carriage return at the end of each record and to be compliant with the type trees of the map.

Note: The following example shows the PARM line to override files names, for example with INFILE instead of REVERSEI, and using the /V parameter at the same time:

```
PARM = REVERSEB /I1 INFILE /VXOD,XOA INFILE /VXOD,XOA REVERSEO
```

All the DDNAMEs in the PARM line must be declared in the execution step of WebSphere Transformation Extender (Figure 6-26).

```
//DTX    EXEC PGM=DTXCMDSV,REGION=0M,
//    PARM='REVERSEB /VXOD,XOA REVERSEI /VXOD,XOA REVERSEO'
//STEPLIB  DD  DISP=SHR,DSN=&DTXLIB
//         DD  DISP=SHR,DSN=&XMLLIB
//*
…….. …(cut) …….
//* The map ddname specified in the JCL must match the map name specified
//* on the command line defined in the PARM statement
//REVERSEB DD  DISP=SHR,DSN=&MAPLIB
//*
…….. …(cut) …….
//*
//* Define the output dataset
//REVERSEO DD  DSN=&MAPOUT,
//             DCB=(RECFM=V,LRECL=80),
//             UNIT=&UNIT,
//             SPACE=(TRK,(1,1),RLSE),
//             DISP=(NEW,CATLG,DELETE)
//*
//*
//* Define the input dataset
//REVERSEI DD  DISP=SHR,DSN=&MAPINPT
//*
```

Figure 6-26 DDNAMEs resolution

For output files, allocation of the file space might be needed if the file must be created at each execution.

Note: If this is the case, by using DISP=(NEW,CATLG,DELETE), the output file must be deleted before each invocation of WebSphere Transformation Extender steplib. See the example of deletion in Chapter 9, “Integration scenario: WebSphere Transformation Extender on System z” on page 623.

The following error message is displayed during the next execution if the file already exists:

```
FOR DATA SET WTX.OUT.REVERSE
IGD17001I DUPLICATE DATA SET NAME ON VOLUME OP1TSA
```

In the JCL, the PARM line can be on one line following the PARM= keyword or on several lines in a file:

- Standard input (up to 100 characters) on one or two lines

Example 6-5 Standard input

```
//WTX   EXEC PGM=DTXCMDSV,
// PARM='NOMMAP /I1 FICIN /VX15 FICIN /I2 LOOKUP /VX15 LOOKUP'
```

- Using inline JCL, PDS, or a sequential file

Example 6-6 Using inline JCL

```
//WTX   EXEC PGM=DTXCMDSV,PARM='/@CMDS'
//*  inline Command file  CMDS
//CMDS    DD *
  NOMMAP /I1 FICIN /VX15 FICIN
  /I2 LOOKUP /VX15 LOOKUP /OI QEMP
  NOMAP2 /TS /O3 FICOUT
/*
//*  end of local command file CMDS
```

Note: Several maps can be launched in sequence, following the order in the PARM command line. See the example in Chapter 9, “Integration scenario: WebSphere Transformation Extender on System z” on page 623.

Location of WebSphere Transformation Extender system files

Most system files are set to SYSOUT. Their DDNAMEs are predefined (Figure 6-27).

```
//* Sysout datasets.  SYSPRINT, SYSOUT and CEEDUMP are required by
//* Language Environment.
//*
//DTXLOG   DD  SYSOUT=&CLASS             Execution log
//DTXAUD   DD  SYSOUT=&CLASS             Audit file
//DTXTRCE  DD  SYSOUT=&CLASS             Trace file
//SYSPRINT DD  SYSOUT=&CLASS
//SYSOUT   DD  SYSOUT=&CLASS
//CEEDUMP  DD  SYSOUT=&CLASS
//*
```

Figure 6-27 Predefined DDNAMEs

DTXLOG contains the WebSphere Transformation Extender version (and build), map results, card settings, number of objects, temporary files defined, and so on.

Note: DTXLOG outputs can be suppressed by using the **/NL** option in the PARM line.

DTXTRCE and DTXAUD are present only if the settings in the map are ON, or if the options **/Txx** or **/Axx** were used in the command line. Options **/T** and **/A** suppress the trace and audit log settings at execution time.

CEEDUMP is displayed in case of execution environment problems.

Temporary data sets

The number of temporary files to create depends on the characteristics of the map itself. It is not always obvious to determine the exact number and the sizes to allocate. For complex maps or for big files, some tests for adjustment might be necessary. Looking at the SYSOUT give a good indication about where to put resources.

Return code 12 with B37 error message: A return code of 12 with a B37 error code means that allocation of an output file or a temporary file does not have enough space:

```
IEC030I
B37-04,IFG0554A,MANTWTX,DTX,SYSTMP02,AF1A,OP1SCR,SYS08270.T092011.RA
000.MANTWTX.TEMP02.H01
 IEF142I MANTWTX DTX - STEP WAS EXECUTED - COND CODE 0012
```

Allocating the temporary files is key for performance (Figure 6-28).

```
/*
//SYSTMP01 DD DSN=&&TEMP01,
//            DISP=(NEW,DELETE,DELETE),
//            DCB=(RECFM=FBS,LRECL=&LRECL),
//            UNIT=&UNIT,
//            SPACE=(TRK,(5,1))
//SYSTMP02 DD DSN=&&TEMP02,
//            DISP=(NEW,DELETE,DELETE),
//            DCB=(RECFM=FBS,LRECL=&LRECL),
//            UNIT=&UNIT,
//            SPACE=(TRK,(5,1))
//SYSTMP03 DD DSN=&&TEMP03,
//            DISP=(NEW,DELETE,DELETE),
//            DCB=(RECFM=FBS,LRECL=&LRECL),
//            UNIT=&UNIT,
//            SPACE=(TRK,(5,1))
//SYSTMP04 DD DSN=&&TEMP04,
//            DISP=(NEW,DELETE,DELETE),
//            DCB=(RECFM=FBS,LRECL=&LRECL),
//            UNIT=&UNIT,
//            SPACE=(TRK,(5,1))
//SYSTMP05 DD DSN=&&TEMP05,
//            DISP=(NEW,DELETE,DELETE),
//            DCB=(RECFM=FBS,LRECL=&LRECL),
//            UNIT=&UNIT,
//            SPACE=(TRK,(5,1))
```

Figure 6-28 Temporary work files

Pre-allocating work files: Pre-allocating work files is not mandatory for small amounts of data, but we highly recommend it for performance reasons.

Use the following rules to set the number of temporary files:

- One temporary data set is allocated in all cases, which is a general use work file.
- One temporary data set is also allocated for each input.
- Depending on how the map is constructed, the map might create a temporary data set for each output to serve as its work file.
- In addition, one temporary data set is always allocated for each input or output defined with an RECFM that is not fixed unblocked (F) or fixed blocked standard (FBS).
- Because you can develop a map with no inputs, and output work files cannot be used in some situations, the minimum number of possible temporary files for a map execution is 1. The maximum possible number is 1 + (2 x number of inputs) + (2 x number of outputs).

Note: In theory, the maximum size of a single work space is 1,073,545,221 bytes. To calculate the space requirements for a particular map, you must determine the maximum data size of each input and each output. A specific input uses the following formula:

```
TSQ size = (max data size x 5)
```

The criteria for the multiplier of 5 is 1 for the shadow file and 4 for the workspace used by the core transformation service. The amount of space that is required by the core transformation service might be less. It depends on a number of factors, such as the number of objects that are being validated. A value of 4 is typical for a complex type tree with many types defined.

See the example on page 16 in the *z/OS Configuration* guide (`1145.pdf` file) in the IBM WebSphere Transformation Extender Online Library.

6.3.5 Submitting a job

Type and send a SUBMIT command from the JCL editor panel to create a new task. When you look in the SDSF panels, the JES2 queues show the job status.

6.3.6 Examining the results

When an execution has ended, look in the output queue to see all execution outputs similar to the example in Figure 6-29.

```
------------------------------------------------------------------------------
 SDSF JOB DATA SET DISPLAY - JOB WTXEX    (JOB15574)     DATA SET DISPLAYED
 COMMAND INPUT ===>                                        SCROLL ===> PAGE
 NP   DDNAME   StepName ProcStep DSID Owner    C Dest              Rec-Cnt Page
      JESMSGLG JES2                 2 WTXUSER  T LOCAL                18
      JESJCL   JES2                 3 WTXUSER  T LOCAL               153
      JESYSMSG JES2                 4 WTXUSER  T LOCAL                57
      DTXLOG   DTX                101 WTXUSER  T LOCAL               108
      DTXTRCE  DTX                103 WTXUSER  T LOCAL               753
```

Figure 6-29 SDSF panel

DTXLOG shows information about the WebSphere Transformation Extender version, the PARM line as interpreted, cards properties, and map settings (Figure 6-30).

```
*** IBM WebSphere Transformation Extender Execution Log for Map REVERSEB
Batch Command Server 8.2.0.3(58)

REVERSEB /VXOD,XOA REVERSEI /VXOD,XOA REVERSEO /TIO

Map REVERSEB has 2 cards (compilation level 7)
Input card 1
  - Card name       : from
  - Source name     : reversei.txt
  - Source type     : Data file
  - Processing Mode : Burst
  Default is ddname REVERSEI
Output card 1
  - Card name       : to
  - Destination name: reverseo.txt
  - Destination type: Data file
  - Processing Mode : Burst
  Default is ddname REVERSEO

Execution options:
DD:REVERSEI will use record separator(s) x0D,x0A
DD:REVERSEO will use record separator(s) x0D,x0A
Memory page size 65536 bytes, page count 8
Created workfiles are on disk
Trace input
Trace output
Validate all input cards
Validate object restrictions
Validate object size
Validate object presentation
Audit report not requested

…
```

Figure 6-30 DTXLOG output

The execution report follows with CPU usage and a return code.

The last part is the “File Usage” section (Figure 6-31). It provides information about files that were used. This information can be useful to estimate the number and size of temporary files.

```
 ( …….)
Burst 5 : elapsed CPU time 0.03 seconds
  Objects found in burst: 9
  Objects built in burst: 13
  Total objects found: 45
  Total objects built: 65

Mapping return code 0, severity NONE:
Map completed successfully
Job step return code is 0
Elapsed CPU time 0.05 seconds
Virtual storage used by engine 3826481 bytes
Exit Statistics:
dllsamp called 10 times

File Usage:
Map file, sysname REVERSEB, dsn BB26159.SDTXSAMP(DTXBMMVS)
 Dev DISK, org PDSMEM, recfm FB blksize 32720 lrecl 80
Work file, sysname SYSTMP01, dsn
SYS08275.T080554.RA000.WTXEX.TEMP01.H01
 Dev DISK, org SEQ, recfm FBS blksize 32760 lrecl 32760
Input 1:, sysname REVERSEI, dsn BB26159.SDTXSAMP(DTXRTEXT)
 Dev DISK, org PDSMEM, recfm FB blksize 32720 lrecl 80
Copy of input 1, sysname SYSTMP02, dsn
SYS08275.T080554.RA000.WTXEX.TEMP02.H01
 Dev DISK, org SEQ, recfm FBS blksize 32760 lrecl 32760 size 0
Input work file 1, sysname SYSTMP03, dsn
SYS08275.T080554.RA000.WTXEX.TEMP03.H01
        ( …….)
Copy of output 1, sysname SYSTMP04, dsn
SYS08275.T080554.RA000.WTXEX.TEMP04.H01
 Dev DISK, org SEQ, recfm FBS blksize 32760 lrecl 32760 size 0
Output work file 1, sysname SYSTMP05, dsn
SYS08275.T080554.RA000.WTXEX.TEMP05.H0
Paging statistics: pages written 5, pages read 0
  Dev DISK, org SEQ, recfm FBS blksize 32760 lrecl 32760
```

Figure 6-31 DTXLOG output

When the trace setting is ON, which is either set during development or with a **/Txx** option in PARM JCL line, the corresponding output is in DTXTRCE. The output in DTXTRCE is exactly the same as in all other platforms.

6.4 Focus on WebSphere Transformation Extender for z/OS

In this section, we discuss the key concepts, actions, and functions of WebSphere Transformation Extender, with a special focus on what is unique within the z/OS environment.

6.4.1 Input data validation: Error handling by using the restart flag and REJECT function

When a map begins execution, the input data must first be validated to ensure that the data for each input conforms to the definition of the corresponding input card. If invalid data is encountered, the map will stop unless you set a restart attribute when defining the input data. Usage of the REJECT function when mapping the output data gives the possibility to get the invalid data.

This mechanism works too for XML input data, using the Xerces parser or a classic type tree. It is not applicable when reading XML with a direct XML Schema Definition (XSD) input card. That is using the XSD directly as a data definition. Because there is no type tree, the RESTART flag cannot be set.

Usage of the Audit log feature permits the capture of validation errors on input, output, and map rules. For example, if some data does not meet the type tree definition (for example, if the type is not in a restriction list), it is displayed in the data log part of the audit log.

On z/OS Batch, the audit log has a predefined DDNAME, which is DTXAUD. This file can be affected to SYSOUT or to any specific data set. It can be used as input to another map that contains more explicit error messages and provides a more user friendly report. It can also be sent through WebSphere MQ to an error handling alert system if the REJECT function was activated.

6.4.2 Reading and writing XML data

When dealing with XML data on z/OS, the following topics are also important:

- Encoding (UTF-8 versus EBCDIC)
- XSD or DTD file location
- Validation, trace, and error handling

Preserving the encoding of XML files

To preserve the encoding of XML files that are created on an ASCII platform, transfer the files (including such files as the Resource Registry .mrn files) in BINARY mode (by using FTP) and not ASCII mode.

Note: These files, transferred in BINARY mode, are still ASCII or UNICODE-encoded. Therefore, they are not directly readable on the mainframe platform.

An XML instance document might contain an encoding attribute that identifies the code page of its content. If you transfer these files to your mainframe in ASCII mode or convert the files to EBCDIC, you must manually change the encoding setting in the resulting XML instance document to identify the correct code page (normally changing UTF-8 to IBM-1047). This change ensures that the XML parser correctly processes the file.

XSD or DTD file location

All files that are used by WebSphere Transformation Extender on z/OS must be declared with a DDNAME. When reading or writing XML data, you can define the data in the input or output card of the map by using one of the following methods:

- Use a type tree imported from an XSD or DTD file by using the Classic option of the importer
- Use a type tree imported from an XSD or DTD file by using the Xerces option
- Use native schema support (XSD file directly)

Native schema support: With the release of V8.2, maps can now use schemas directly as definitions in input or output cards. In previous releases of WebSphere Transformation Extender, if you wanted to use schemas with the product, you were required to use the XML Schema Importer to create type trees. To use this feature, select a .xsd file instead of a .mttt file from within the Map Editor.

How the XML data is defined has a consequence on the z/OS files implementation. To illustrate and demonstrate these features, we use the schema file shown in Figure 6-32. This example corresponds to a schema for the XML file that is produced in Chapter 9, “Integration scenario: WebSphere Transformation Extender on System z” on page 623. A namespace definition has been added to be more compliant with complex schemas that are found, for example, in the SEPA industry pack.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.example.org/NewOrders"
        elementFormDefault="qualified"
        xmlns="http://www.example.org/NewOrders">
  <xs:element  name="File" type="File" />
  <xs:complexType name="File">
      <xs:sequence>
        <xs:element  minOccurs="0" maxOccurs="unbounded" name="NEWORDERRECORD">
          <xs:complexType>
            <xs:sequence>
              <xs:element id="x5" name="CUSTOMER_ID" type="xs:string" />
              <xs:element id="x6" name="SHIP_TO_CODE" type="xs:string" />
              <xs:element id="x7" name="ORDER_DATE" type="xs:string" />
              <xs:element id="x8" name="WANT_DATE" type="xs:string" />
              <xs:element id="x9" name="CATALOG_NUMBER" type="xs:string" />
              <xs:element id="x10" name="QUANTITY_ORDERED" type="xs:integer" />
              <xs:element id="x11" name="UOM" type="xs:string" />
              <xs:element id="x12" name="UNIT_PRICE" type="xs:integer" />
              <xs:element id="x13" name="EXTENDED_PRICE" type="xs:integer" />
              <xs:element id="x14" name="RECEIVED_DATE" type="xs:string" />
              <xs:element id="x15" name="ROUTING_INFO" type="xs:string" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Figure 6-32 XSD example

Using a Classic type tree for XML

The schema file is imported by using the Design Studio import function, with the Classic option (see Figure 6-33).

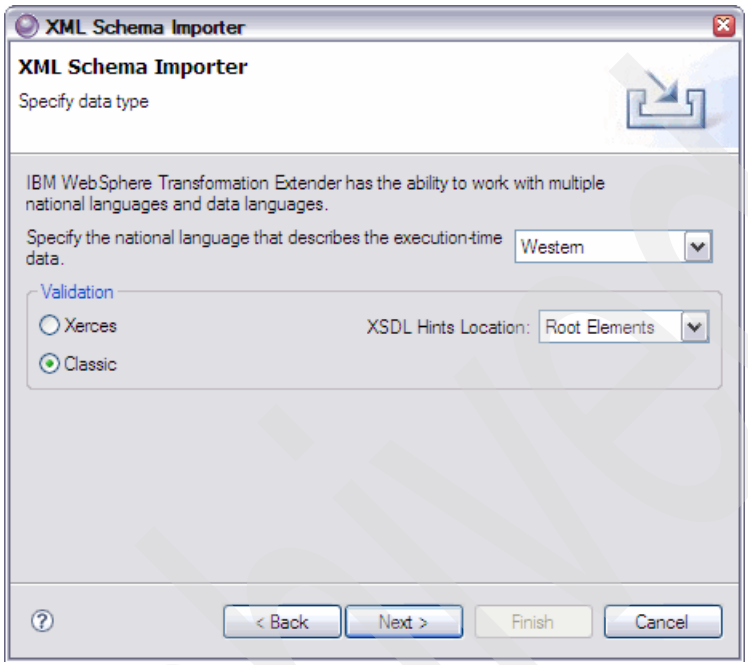

Figure 6-33 Importing XSD with the Classic option

There is no difference with a standard type tree, and there is no extra file to declare in the JCL. The XSD is not required at execution time. All validations are done against the type tree definitions.

Using a Xerces type tree

The XSD schema file is imported with the Xerces option. (To keep namespace declarations in output files, the Root element for XSDL has been selected.) When deploying the map, the XSD file must be uploaded and associated, because it is used for parsing and validation.

Note: Upload the XSD (DTD) file in binary mode because the expected encoding is UTF-8.

In the JCL, this file must be declared with a DDNAME. WebSphere Transformation Extender uses that DDNAME to locate the XSD file as shown in Example 6-7.

Example 6-7 JCL being declared with a DDNAME

```
// SET MAPINPT=WTX.MAPS(NORD2P4)
// SET MAPOUT=WTX.OUT.PROG4
// SET MAPXSD=RTC1.PACS.BINARY.XSD
(...)
//NORDXSD DD DISP=SHR,DSN=&MAPXSD
//XMLIN    DD DISP=SHR,DSN=&MAPINPT
//XMLOUT   DD DISP=SHR,DSN=&MAPOUT
//PARMS    DD *
 NOMMAP /I1 XMLIN /O1 XMLOUT
```

The problem is now that you must associate the XML definition in WebSphere Transformation Extender with this DDNAME. If you do nothing to handle the schema file location, WebSphere Transformation Extender issues an error message in the DTXTRCE file when the trace setting is on (Figure 6-34). This message means that the external Xerces parser used the Location property set for Windows system.

```
TYPE X'0002' (Warning (2), "XMLParser: XML data is valid but
warning(s) occurred.
SAXParseException, Warning Ýline: 0 column: 0~ An exception
occurred§ Type:RuntimeException, Message:Warning: The primary
document entity could not be opened. Id=NewOrders.xsd
0Error (-1), "XMLParser: Input XML data is invalid."
 SAXParseException, Error Ýline: 2 column: 49~ Unknown element
'File' 0).
0External Parser returned Error
```

Figure 6-34 Error message when the XSD file is not a known XML parser

Three methods are available to associate a WebSphere Transformation Extender input or output card with the XSD DDNAME:

- Change the type tree that defines the card if it is a Xerces generated type tree
- Change the schema location in the card definition of the map
- Use the M parameter in a command line

We explain each of these methods in the sections that follow.

Changing the type tree when using a Xerces imported type tree

When dealing with an XML type tree generated with the Xerces option, change the Location from the XSD Windows-like file name to a //DD Statement. Select the root element in the type tree, and select **Intent** → **Validate as** to open the properties (see Example 6-8 and Example 6-9).

Example 6-8 Original to change

```
http://www.example.org/NewOrders  NewOrders.xsd
```

Example 6-9 illustrates how the location value (NewOrders.xsd in Example 6-8) has changed from a Windows-type file name to a //DD statement. The result from the change is //DD:NORDXSD.

Example 6-9 Final text for Location in the type tree

```
http://www.example.org/NewOrders     //DD:NORDXSD
```

In this example, NORDXSD is the DDNAME that is used in the JCL for the schema file as shown in Figure 6-35.

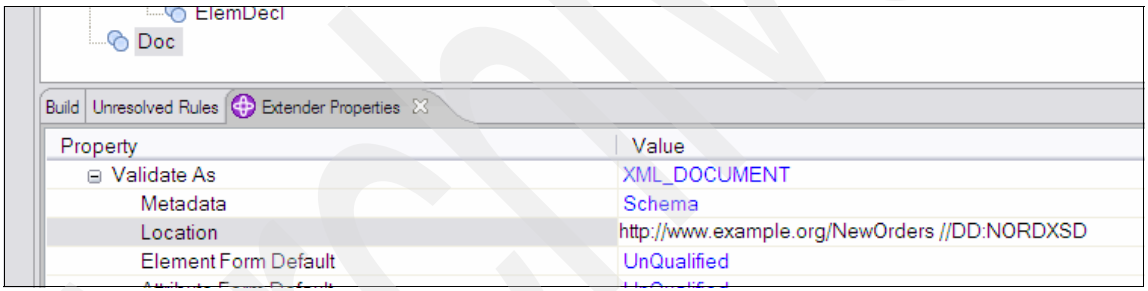

Figure 6-35 Defining a DDNAME as the location of an XSD file in a type tree

More information: For information about coding a URI to access an MVS data set, see Chapter 1 in the *IBM XML Toolkit for z/OS Users Guide*, SA22-7932-06, which you can find at the following Web address:

`http://www-03.ibm.com/servers/eserver/zseries/software/xml/pdf/ug_v1r9.pdf`

With this method, there is one type tree for Windows unit testing and one type tree for z/OS execution. Therefore, there is a map for Windows and a map for z/OS.

Changing the schema location in a card definition of the map

This method is valid for Xerces generated type trees and for native XSD usage. However, this means a map for Windows and a map for z/OS. For example, for a native XSD input card, change the value of the Metadata property in the card settings (Figure 6-36). You must do the same when using Xerces generated type trees.

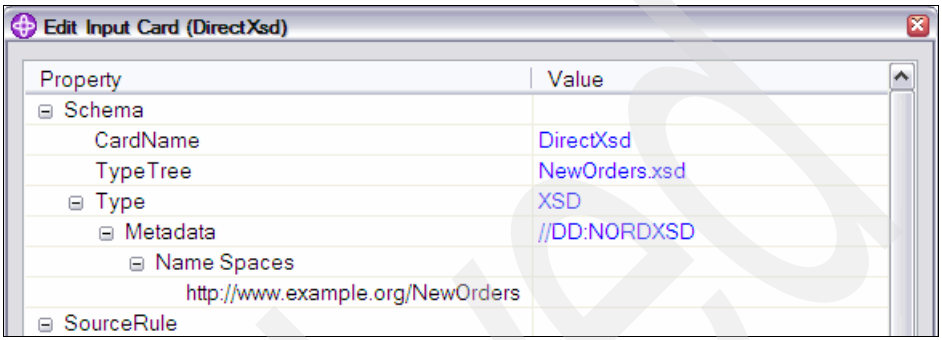

Figure 6-36 A DDNAME as the location of XSD file in a card metadata property

Note: The MPIP_CARD_METADATA_LOCATION card object property is used with the WebSphere Transformation Extender API.

Using the M parameter in a PARM option line in JCL

This method does not change the original type tree nor map. It is used in the Command Server PARM line at execution time (Figure 6-37).

```
000030 //*  ------------ WTX COMMAND SERVER
000031 //DTX   EXEC PGM=DTXCMDSV,REGION=0M,
000034 //*  Option : use M parameter
000035 //   PARM='MAPREF  /I1M ''/DD:MAPXSD'' /VX15 XMLERROR'
000036 //STEPLIB  DD  DISP=SHR,DSN=&DTXLIB
000037 //         DD  DISP=SHR,DSN=&XMLLIB
000038 //*
```

Figure 6-37 /IM option in the PARM option line

Syntax for z/OS: Use double quotation marks (" ") if you are using a single line PARM, or use single quotation marks (' ') when using parameters in a command file. Note that the space after **/I1M** is mandatory. This option is independent of overriding the file name. Consider the following example when using both options:

```
//   PARM='MAPREF /I1 MYFILE /I1M "/DD:MAPXSD" /VX15 XMLERROR'
```

Using native XSD

All the proceeding methods to override the schema file name with a DDNAME work for schema files that have no includes. Only the first level can be addressed with the properties or the M parameter.

In case of embedded schema files, you must change all include statements and give a DDNAME for each `Schemalocation` instead of the Windows-like file name with a .xsd extension. The syntax is `Schemalocation = "//DD:DDNAME"`.

For example, in the provided ipo.xsd example, change the following lines:

```
<!-- Include address constructs -->
<include schemaLocation=”address.xsd”/>
```

Change the lines as follows:

```
<!-- Include address constructs -->
<include schemaLocation=”//DD:ADDRESS”/>
```

In the JCL, ADDRESS is the DDNAME for the XSD file.

Input data validation

When using the restart flag in an input type tree and the GETXMLERRORMSG() function in an extra output card, all errors can be written in a file, with some possible extra information defined in the card (Figure 6-38).

```
Map Version   1.00
Input File:DD:XMLORDER
Process Date:080930
Time:10:38:54
Errors  in file   : Error (-1), "XMLParser: Input XML data is
invalid."
SAXParseException, Error Ýline: 11 column: 29~ Datatype error:
Type:InvalidDatatypeValueException, Message:Value '7OA' does not
match regular expression facet '[+\-]?[0-9]+''
```

Figure 6-38 Output produced by an Error card

This function is useful because it gives all errors in the input file with a line and column number for each error.

Choosing between a Classic XML tree, Xerces XML tree, and Direct XSD

The options of the Classic XML tree, Xerces XML tree, and Direct XSD are not interchangeable. You must decide on one option in the beginning. After a map is developed with one option, to change the method, you must re-do the mapping.

To summarize the different options, review the following pros and cons.

- Pros:
  - Classic generated type tree
    - There is no schema or DTD file to deploy.
    - Normal XML tags can be changed manually because they are standard initiator and terminator properties.
    - It can be used for mixed data format, mixing XML and others (for example, reading alternatively BLOBs and embedded XML data)
  - Xerces type tree
    - The input validation with all errors in the input file can be shown in a file.
  - Direct XSD
    - There is no type tree generation.
- Cons:
  - Classic generated type tree
    - This method is slower at execution.
    - There are no user friendly error messages.
  - Xerces type tree
    - You must deploy a schema or DTD file with the map.
  - Direct XSD
    - Error diagnostics do not have a restart or REJECT mechanism, which means that there are no GETXMLERRORMSG function outputs.

6.4.3 Packed decimal data

Packed decimal data is a binary representation of numbers. WebSphere Transformation Extender has built-in mechanisms to manage these types. For example, in COBOL copybook, they are written as shown in Figure 6-39.

```
10 NUM2                    PIC S9(08) COMP.
10 NUM3                    PIC S9(13)V9(2) COMP-3.
```

Figure 6-39 COBOL copybook

Figure 6-40 shows the properties in the imported type tree.

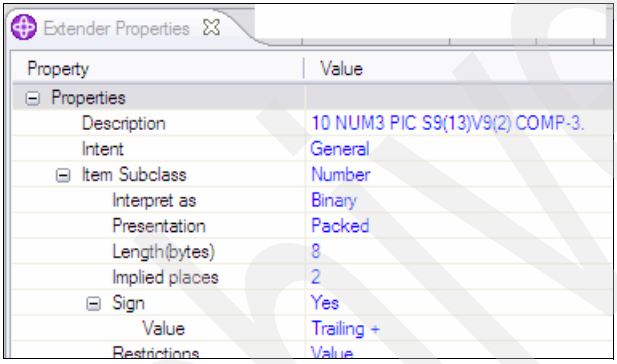

Figure 6-40 Packed decimal data

Attention: When moving files with packed decimal data, do not change the encoding.

6.4.4 Code page conversion fallback

The CodePageFallback map setting is used to specify, on each map, the code page conversion fallback behavior when mapping between objects with a specific data language set, and when an unknown character is found during the conversion process.

Example 6-10 shows a sample of data with unknown characters.

Example 6-10 Original data with UTF-8 encoding

```
abcù%¤é¼@"ÃœÃŸ?????Ï?Î0à@\±
Redbook
WTX
è£¤??ÃŸcba
```

The CodePageFallback map settings have the following options (Figure 6-41):

- SUBSTITUTE for using the International Components for Unicode (ICU) substitution character that is specified in the conversion tables
- SKIP for skipping characters that cannot be converted to the target code page

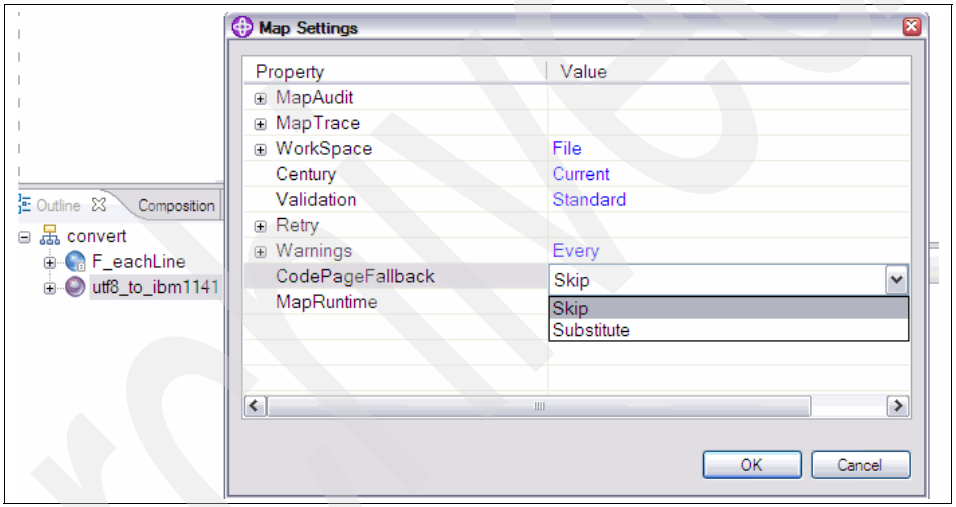

Figure 6-41 Map CodePageFallback settings

Also you can use a command line override in either of the following ways:

- `/S` to specify the skip command option
- `/S!` to specify the substitution command option

Neither of these options is yet available on the System z platform.

Example 6-11 shows the result of using the SKIP option. In Example 6-11, the Euro symbol does not exist in the target code page. It is ignored, and therefore, the line is shorter.

Example 6-11 Result using SKIP option

```
******************************** Top of Data ***********************
abc%@ÜßO@\                                          .
Redbook.
WTX.
ßcba.
******************************** Bottom of Data ********************
```

Note: When using the SKIP option in a map that produces an output with fixed-length type data, you must have the “padding” property in each field. Because the output might be shorter text, you must keep the length of the field the same.

In Example 6-12, using the SUBSTITUTE option preserves the length of the text.

Example 6-12 Result using SUBSTITUTE option

```
****************************** Top of Data ****************************
abc.%..@.Üß.......O.@\.                                        .
Redbook.
WTX.
...ßcba.
****************************** Bottom of Data ************************
```

6.4.5 The convert function

Sometimes the CodePageFallback mechanism does not fit all requirements, and the substitution character cannot be used. There are some cases where the ICU substitution default character itself is not known by the destination code page. The convert map function can help solve this problem. This function uses a transcodification table that can be filled by any characters and stored in a file.

The convert map function must be used for each field to convert. This means a heavier mechanism than the CodePageFallback mechanism, witch is at the map settings level.

Figure 6-42 shows a sample map using the convert function with WebSphere Transformation Extender in the `<installdir>\examples\general\portdata` directory.

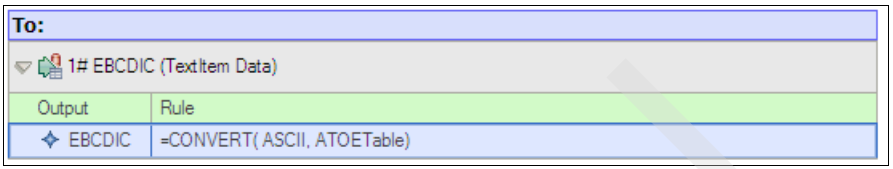

Figure 6-42 WebSphere Transformation Extender portdata sample

6.4.6 Zero execution return code in JCL

Many production systems do not allow tasks to continue if there is a nonzero return code after execution. WebSphere Transformation Extender might issue warnings (RC=04) for some errors such as length of data or when using the restart-REJECT mechanism (Figure 6-43). If these situations are handled by WebSphere Transformation Extender or another application, a return code of zero is desired.

```
09.07.31 JOB15371 $HASP165 WTXSCEN3 ENDED AT WTSC59 MAXCC=4 CN (INTERNAL)
***
```

Figure 6-43 Error code example

Several cases can be handled with the PARM line options as explained in the following sections.

Ignore Validation Options (/G)

Use the Ignore Validation Options execution command (**`-G`**) to specify the item properties, if any should be ignored during validation.

The different options of **`/G[R][P][S][T]`** change the behavior of WebSphere Transformation Extender:

[R] Ignore restrictions for all items that are displayed in the input data.

[P] Ignore the presentation settings of items.

[S] Ignore the minimum and maximum size of items that are displayed in the delimited data.

[T] Ignore restrictions for all items that are displayed in the input data with no warnings.

Ignore Warnings (/Z)

The execution Ignore Warnings execution command (**-Z**) specifies, for a map, selected warning codes that should be ignored and a return code of that should be issued.

Important: Using this command impacts the OnSuccess settings, OnFailure settings, and so on. When using the **/Z** command, you cannot specify the same warning code by using the Fail on Warnings command (**/F**).

6.4.7 The run map function

The **run map** function is used in a mapping rule to call another map. On z/OS, the name of the called map must be a DDNAME. Another consideration is the encoding of data when transmitting the data to or from the called map. Example 6-13 and Figure 6-44 shown an example of using this function.

Example 6-13 Run function example

```
RUN (“MYMAP”,”OF1 “+ “SEPA” + NUMBERTOTEXT(index) + “XML”+
HANDLEIN(1,In1))”
```

MYMAP must be defined as a DDNAME in the JCL (Figure 6-44).

```
//MYMAP  DD DSN=WTX.MAPS(MYMAP), DISP=SHR
```

Figure 6-44 JCL declaration for called map DDNAME

In Figure 6-45, the output file is based on the index of the input record. For example, the first set of input records is 1, the second set is 2, and so on. The full names of output files are SEPA1XML, SEPA2XML, SEPA3XML, and so on. This means that the JCL should define all possible files.

```
//SEPA1XML DD DSN='RTC1.SEPA1.XML',DISP=SHR
//SEPA2XML DD DSN='RTC1.SEPA2.XML',DISP=OLD
//SEPA3XML DD DSN='RTC1.SEPA3.XML',DISP=(NEW,….)
…etc
```

Figure 6-45 Declaring all DDNAMEs for files that are used in the run map function

In the following section, we show a mechanism that can simplify the declaration of DDNAMEs in JCL.

Map name conversion: z/OS Batch Command Server converts the map name used in the RUN function to a legal z/OS DDNAME. It does this conversion by removing drive, path, and file extension information if it exists, converting _ characters to # characters, and translating the remaining characters to uppercase. For example, the map rule `RUN(\ install_dir \mymaps\map_10.mmc~)` executes the map that is identified by the DDNAME MAP#10.

6.4.8 Dynamic DD references with a PDS (member) mechanism

Some maps can call several other maps. A good example is the validation map in the SEPA pack, which can call different maps depending on the nature of data. The called maps use several input files and different output files.

A situation can result where dozens of DDNAMEs must be declared in the same JCL, which making it difficult to read and maintain. One way to solve this for maps is to have all maps as a member of a PDS. In the JCL, only the name of this PDS is declared. Figure 6-46 shows how the RUN function is used.

```
RUN (MYDDNAME(MYMAP1)
.....
RUN ( MYDDNAME(MYMAP2)
.....
```

Figure 6-46 Maps called from a PDS

The DDNAME “MYDDNAME” is associated with only the PDS, and members MYMAPO1, MYMAP2 are dynamically known at execution time.

For files, the problem cannot always be solved this way because properties of files can be different. It can be applied with files that have the same properties (RECFM, LRECL….) and be in the same PDS. This is the case for XML files that are uploaded in binary mode.

The problem remains in the case of numerous output files that are generated, for example, in a loop during mapping. If the number of expected file is large, a solution is to have a C or COBOL dynamic link library (DLL) that is called by the EXIT function that allocates the files.

6.4.9 CPACKAGE and CTEXT functions in the RUN, GET, and PUT functions

There are situations where the code page of the data is not the default of the platform where the map runs and where the original data code page must be kept.

For example, consider a case where a map is running on the UNIX System Services (EBCDIC) execution environment with some XML/UTF-8 input data and a rule using the PACKAGE function. In a RUN function, for example, the map converts the data to the NATIVE code page, which is EBCDIC, which might be invalid for the next map to run or for an adapter.

To avoid this problem and keep the original data code page, use the CPACKAGE function instead of the PACKAGE function. Also, specify the character set as NATIVE for the second argument on the rule as shown in the examples in Figure 6-47 and Figure 6-48.

Change the rule from the line shown in Figure 6-47.

```
RUN("MYMAP",.....PACKAGE(GROUP:RECORD).....
```

Figure 6-47 PACKAGE function used

Change the rule to the line shown in Figure 6-48.

```
RUN("MYMAP",.....,CPACKAGE(GROUP:RECORD, NATIVE).....
```

Figure 6-48 CPAKAGE function to preserve encoding

In this example, when you specify NATIVE in the second argument of the CPACKAGE (or CTEXT) function, the function treats the data as though it is already in the appropriate code page, and therefore, does not attempt to convert it.

6.4.10 The z/OS DB2 adapter

Three methods can be used with the z/OS DB2 adapter:

- Method 1: Direct DB2 call (Figure 6-49)

```
DBQuery (“Select * from mytable”, “mydatabase.mdq”, “myDB”)
```

Figure 6-49 Direct DB2 call

  - Pros
    - The .mdq file is compiled into the map. Therefore, there is no need to copy the .mdq file separately to the running environment.
    - The user ID and password are hidden from both the source and compiled map.
  - Con
    - To override the user ID and password, the .mdq file must be reconfigured and the map must then be recompiled. Configuration is more static.
- Method 2: Using a .mdq file (Figure 6-50)

```
DBQuery (“Select * from mytable”, “-MDQ mydatabase.mdq -DBNAME
myDB”)
```

Figure 6-50 DB2 call with external MDQ

  - Pro
    - The .mdq file (created in the Database Interface Designer; see 5.3.8, “Database Interface Designer” on page 296) can be changed without recompiling the map.
  - Cons
    - Each time DBQuery is run, the map parses the .mdq file to look for database parameters such as user ID and password. One test run shows an elapsed time of three times more than other methods when DBQuery is called many times.
    - The .mdq file is copied separately to the running environment.
    - You must add a DD card in the JCL for the .mdq file (Figure 6-51).

```
DBQuery ("Select …”, -MDQ DDNAME ..).
```

Figure 6-51 Declaring the DDNAME for a .mdq file

- Method 3: DB2 call with all explicit parameters (Figure 6-52)

```
DBQuery("Select * from mytable", "DBTYPE db2 -SOURCE myodbc -USER
db2admin -PASSWORD mypasss")
```

Figure 6-52 DB2 call with all parameters explicit

  - Pros
    - This method is more dynamic. The user ID and password are changed on the fly.
    - No .mdq file is needed.
    - In the following Cons list, see the second point regarding the Call Interface method.
  - Cons:
    - The user ID and password are exposed in the map.
    - If running the map in a z/OS environment (Example 6-14) by using the Call Interface method (as opposed to ODBC), this problem goes away because the user ID is passed from the batch job based on the person who submits the job.

Example 6-14 Method 3 in z/OS environment

```
DBQuery("Select * from mytable", -DBTYPE DB2MVS -PLAN DBUTILE
-SUBSYTEM ")
```

The recommended method is to use either method 1 (direct DB2 call) or method 3 (DB2 call with all parameters explicit).

6.4.11 The EXIT function

There are special considerations when using the EXIT function with the z/OS Batch Command Server. See the *Command Server* guide (`1010.pdf` file) and the *Functions and Expressions* guide (`1006.pdf` file) at the following address:

`ftp://ftp.software.ibm.com/software/websphere/integration/wdatastagetx/`

A user exit is invoked by calling the EXIT function in a mapping rule. The EXIT function expects three arguments:

If the user exit program is a DLL, the first argument must contain the name of the DLL load module, and the second argument must contain the name of the function within the DLL.

If the user exit program is *not* a DLL but a program, the first argument must contain a value that is not a null string, for example one or more spaces, and the second argument must contain the name of the load module.

The third argument is a text item. It contains parameter information that is passed to the exit program.

Looking again at the REVERSE map, the first use of EXIT is to call a program called DTXPBSP with an argument LINE (Figure 6-53).

```
=EITHER (EXIT ("", "dtxtpbsp", "LINE"), "Exit DTXTPBSP not found")
```

Figure 6-53 REVERSE user exit rule calling a program

The second use of EXIT (Figure 6-54) is to call a function named “dllsamp” in a DLL file named “dtxdlsmp.” The function has a text as argument, which is the concatenation of three words. The function returns the text data length followed by the length of the three words.

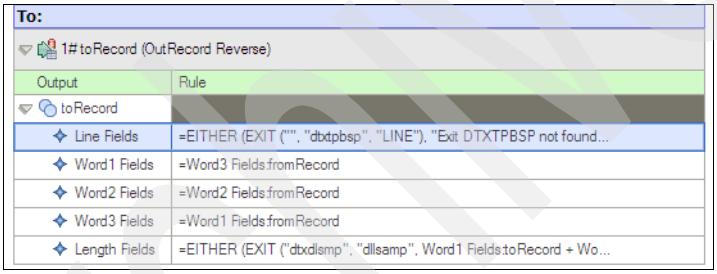

Figure 6-54 User EXIT example

To use these programs and DLL, WebSphere Transformation Extender Command Server must know where they are located. Line 55 in Figure 6-55 adds the library of these modules.

```
000051 //DTX   EXEC PGM=DTXCMDSV,REGION=0M,
000052 //  PARM='REVERSEB /VXOD,XOA REVERSEI /VXOD,XOA REVERSEO /TO1'
000053 //STEPLIB  DD  DISP=SHR,DSN=&DTXLIB
000054 //         DD  DISP=SHR,DSN=&XMLLIB
000055 //          DD  DISP=SHR,DSN=WTX.COBOL.LOAD
```

Figure 6-55 Adding a library to JCL

6.4.12 VSAM considerations

The z/OS Batch Command Server supports the use of VSAM key-sequenced (KSDS), entry-sequenced (ESDS), and relative record (RRDS) data sets for both inputs and outputs. The inputs and outputs are subject to VSAM data sets being predefined and, if used for input, preloaded by using IDCAMS or some other utility.

The z/OS Batch Command Server opens input and output VSAM files for both reading and writing. When used as *input*, all VSAM data sets are processed sequentially. There is no direct access. All records are read. When used as *output*, KSDS files are updated, and ESDS and RRDS files are appended.

6.4.13 The Resource Registry

A *resource name* is an alias that has different values that are defined for multiple deployment environments. A resource value can be either a command card setting for a map, an adapter command override setting for a map or system component, or an argument to environment functions within a map rule (RUN, PUT, GET, DBLOOKUP, DBQUERY, and EXIT).

In addition, the following resources can have an alias:

- Backup files
- Audit files and locations
- Trace files and locations
- Work space locations
- External parser location (defined in the Type Tree Editor)

Resource names that are created in the Resource Registry can be used in maps that run on z/OS platforms (Figure 6-56).

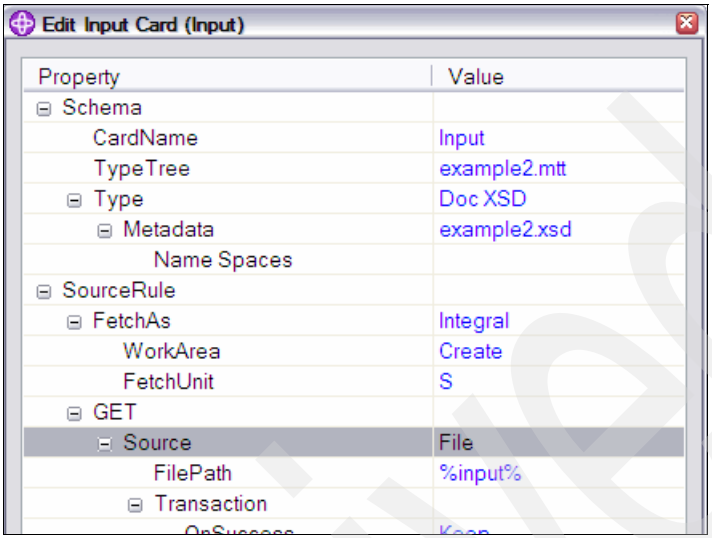

Figure 6-56 Giving an alias to file name

In this example, the alias `%input%` is replaced at run time by the corresponding value in the table of the virtual server, which represents the platform.

The following rules are necessary to use resource names in maps:

- The virtual server that is defined for the z/OS platform must be named MVSSERVER (uppercase, case sensitive; Figure 6-57).

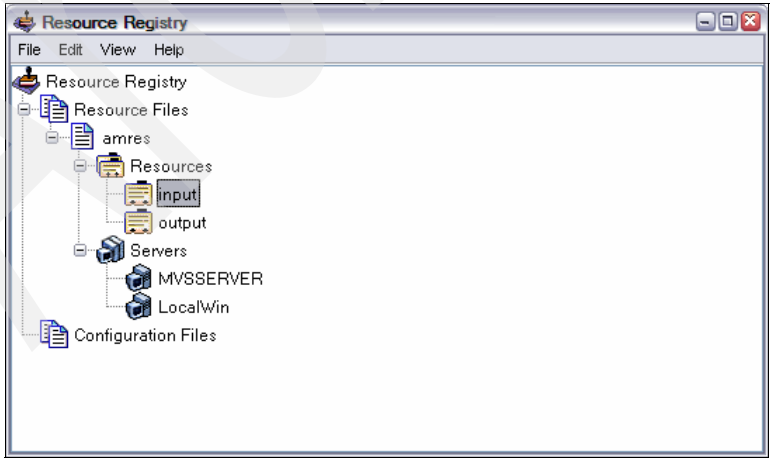

Figure 6-57 Resource Registry Tool - z/OS virtual server

Resource values must conform to z/OS rules for their types, such as DDNAMES and database names. In the example in Figure 6-58, the alias `%input%` takes the value `example2.xml` for LocalWin (Windows platform) and `CARDIN2` for MVSSERVER (z/OS).

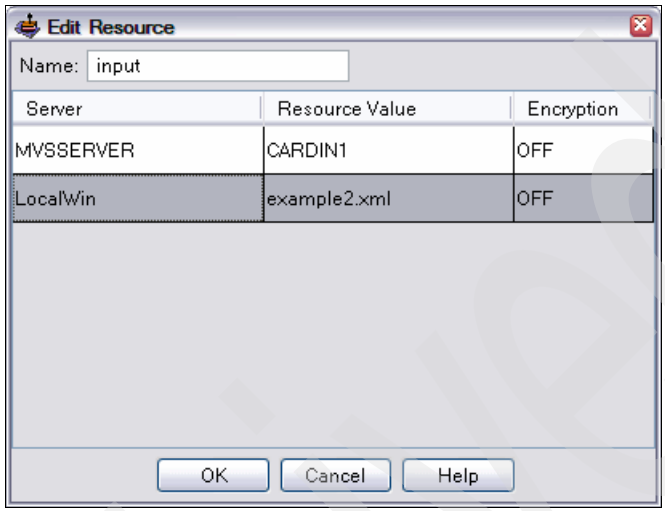

Figure 6-58 Using the Resource view for the %input% alias

By using the MVSSERVER view, all defined resources are listed for this server (Figure 6-59).

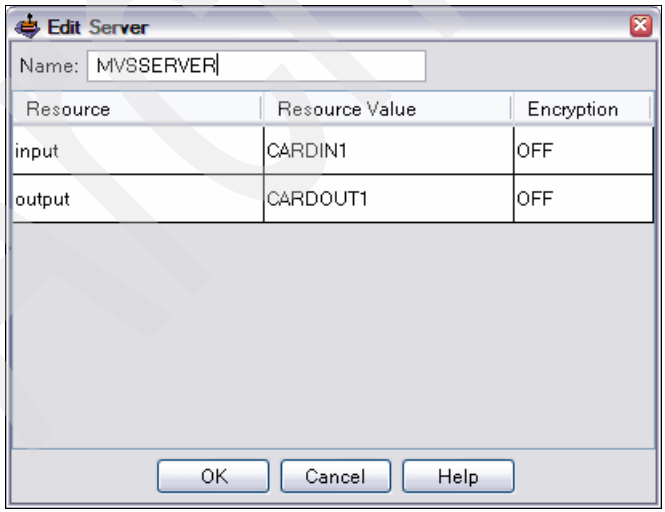

Figure 6-59 MVSSERVER Server view

- The resource name file (.mrn) must be uploaded to the z/OS platform in binary mode (an xml file).
- The name and location of the resource name file are specified in either DTXMRN:
  - Use DDNAME for the Batch region or the DTXMRN VSAM data set definition for the CICS region.
  - In a JCL, add the DDNAME DTXMRM definition for the resource file (Figure 6-60).

```
000041 //*
000042 //*   Resource file
000043 //DTXMRN  DD  DISP=SHR,DSN=WTX.MRM.RES
000044
```

Figure 6-60 JCL declaration for a Resource Registry file

- The resource configuration file (.mrc) is not required.

The Resource Registry file is managed with the development tool Resource Registry utility on the Windows platform. See 5.3.12, “Resource Registry” on page 332, for more details.

6.4.14 The DTXPAGE utility

The DTXPAGE utility calculates suggested settings for memory page size and count for maps.

More information: For documentation for DTXPAGE, see the *Utility Commands* guide (1118.pdf file) at the following address:

ftp://ftp.software.ibm.com/software/websphere/integration/wdatastagetx/

To use the DTXPAGE utility command, make a copy of the JCL file that executes the map on which the DTXPAGE utility command will perform its calculations. Change the program name from DTXCMDSV to DTXPAGE (Figure 6-61).

```
000048 //*
000049 //DTX   EXEC PGM=DTXPAGE,
000050 //  PARM='REVERSEB /VXOD,XOA REVERSEI /VXOD,XOA REVERSEO /TO1'
000051 //STEPLIB  DD  DISP=SHR,DSN=&DTXLIB
000052 //         DD  DISP=SHR,DSN=&XMLLIB
```

Figure 6-61 Changing the JCL for DTXPAGE

The report is written to an output that is specified on the SYSPRINT data definition (DD) statement (Figure 6-62).

```
NP   DDNAME   StepName ProcStep DSID Owner    C Dest              Rec-Cnt
Page
      JESMSGLG JES2              2 MANTEAU  T LOCAL                 19
      JESJCL   JES2              3 MANTEAU  T LOCAL                158
      JESYSMSG JES2              4 MANTEAU  T LOCAL                 60
      SYSPRINT SO              102 MANTEAU  T LOCAL                  7
 s    SYSPRINT DTX             106 MANTEAU  T LOCAL                 23
```

Figure 6-62 SYSPRINT report location

DTXPAGE tests different values for size and number of memory pages and then displays the one with the lowest execution time (Figure 6-63).

```
******************************** TOP OF DATA ************************
IBM WebSphere Transformation Extender
Page Setting Assistant for Maps - version 8.2(58)
  Copyright 2008 IBM Corporation. All rights reserved.
.size  128 count   16 time 689
.size   64 count   16 time 668
.size   32 count   16 time 660
.size   16 count   16 time 624
.size    8 count   16 time 618
.size    4 count   16 time 629
.size    8 count   16 time 629
.size    4 count   16 time 633
.size    8 count   16 time 633
.size    8 count   16 time 639
.size    8 count   20 time 630
.size    8 count   24 time 627
.size    8 count   28 time 636
.size    8 count   24 time 625
.size    8 count   20 time 621
.size    8 count   16 time 640
.size    8 count   20 time 622
.size    8 count   24 time 621
.size    8 count   28 time 627
Optimal page settings for REVERSEB are SIZE: 8  COUNT: 16

******************************** BOTTOM OF DATA ************************
```

Figure 6-63 Output for DTXPAGE

The given values can be used with **/P** option in the PARM line (Figure 6-64).

```
//DTX   EXEC PGM=DTXCMDSV,REGION=0M,
//  PARM='REVERSEB /VXOD,XOA REVERSEI /VXOD,XOA REVERSEO /P8:16'
//STEPLIB  DD  DISP=SHR,DSN=&DTXLIB
//         DD  DISP=SHR,DSN=&XMLLIB
```

Figure 6-64 /P option in the PARM command line

6.4.15 The DTXPROF utility

The DTXPROF utility command is available with the z/OS Batch Command Server. It profiles maps and analyzes map execution behavior.

More information: For documentation about DSTPROF, see *Utility Commands* guide (1118.pdf file) at the following address:

ftp://ftp.software.ibm.com/software/websphere/integration/wdatastagetx/

With a large amount of data, the profiler output can be large or unmanageable. For this reason, profile only a subset of the original data. Reduce the profiler output by specifying a time limit so that nothing is reported until it exceeds the time limit that you specify.

The DTXPROF utility command has the options for the syntax:

- **-f**, **-fs**, **-fx** for the time of functions: all, summary, and above x milliseconds
- **-t**, **-ts**, **-tx** for the time of type functions: all, summary, and above x milliseconds

To use the DTXPROF utility command:

1. Make a copy of the JCL file that executes the map on which the DTXPROF utility command will profile.
2. Change the program name from DTXCMDSV to DTXPROF.
3. Add the specific option to use with the DTXPROF utility command in the PARM field on the EXEC statement of the JCL (Figure 6-65).

```
000048 //*
000049 //DTX   EXEC PGM=DTXPROF,
000050 // PARM='-f -t -o DD:P -dtx "REVERSEB /VXOD,XOA REVERSEI /VXOD,XOA REVE1
000051 //           RSEO"'
000052 //STEPLIB  DD  DISP=SHR,DSN=&DTXLIB
000053 //         DD  DISP=SHR,DSN=&XMLLIB
```

Figure 6-65 Execution step for DTXPROF

4. Add DDNAME for the outputs of DTXPROF, given by the **-o** option (Figure 6-66).

```
000084 //SYSPRINT DD  SYSOUT=&CLASS
000085 //SYSOUT   DD  SYSOUT=&CLASS
000086 //CEEDUMP  DD  SYSOUT=&CLASS
000087 //P        DD  SYSOUT=&CLASS
000088 //*
```

Figure 6-66 DDNAMEs for DTXPROF utility

Note: The PARM line is limited to 100 characters, which is larger than a simple line. This example shows how to use a second line:

- Put a 1 in column 72 of the text (80th of the screen).
- Start the second line with the double forward slash // symbol and end the text at column 16.

If the PARM field length exceeds the 100 character limit, place the DTXCMDSV map command line in a file and specify **-dtxcmdsv?-@ddname?** in the PARM line along with the DTXPROF command line that you want to use.

Consider the following example:

```
//STEP1 EXEC PGM=DTXPROF,
// PARM='-f -t -fs -ts -o DD:P -dtxcmdsv "-@CMD"'
//* Command line for WTX:
//CMD DD *
REVERSE /VXOD,XOA REVERSEI /VXOD,XOA REVERSEO
/*
//P       DD SYSOUT=*
```

Figure 6-67, Figure 6-68, and Figure 6-69 on page 411 show the results of execution.

```
NP   DDNAME   StepName ProcStep DSID Owner    C Dest              Rec-Cnt Page
      JESMSGLG JES2                2 MANTEAU  T LOCAL                19
      JESJCL   JES2                3 MANTEAU  T LOCAL               161
      JESYSMSG JES2                4 MANTEAU  T LOCAL                62
      SYSPRINT SO                102 MANTEAU  T LOCAL                 7
      DTXLOG   DTX               103 MANTEAU  T LOCAL               114
      SYSPRINT DTX               106 MANTEAU  T LOCAL                 3
      P        DTX               109 MANTEAU  T LOCAL                17
```

Figure 6-67 JES2 SYSOUT

```
******************************** TOP OF DATA ***************************
IBM WebSphere Transformation Extender
Map Profiler version 8.2(58)
  Copyright 2008 IBM Corporation. All rights reserved.

****************************** BOTTOM OF DATA **************************
```

Figure 6-68 Content of SYSPRINT DTX

```
******************************** TOP OF DATA ***************************
WebSphere Transformation Extender 8.2(58) Map Profiler results
  Copyright 2008 IBM Corporation. All rights reserved.
Thu Oct  2 07:28:23 2008
Profiling for REVERSEB
Total entries: 170
Total map sets: 1
  Function summary
Function        Iterations            Time
-----------------------------------------
VALIDATE_CARD            5          34
BUILD_CARD               5          29
TYPE_NAME               70           1
STRING_LITERAL          40           0
ADD_OP                  20           0
MAP_NAME                10           6
EITHER                  20           3
EXIT                    20           3
******************************** BOTTOM OF DATA ************************
```

Figure 6-69 Content of P - Outputs of DTXPROFS

Using the **-f** and **-t** options instead of the **-fs** option provides more details. Figure 6-70 shows details of the DTXPROF options.

```
000049 //DTX   EXEC PGM=DTXPROF,REGION=0M,
000050 // PARM='-f -t -o DD:P -dtx "REVERSEB /VXOD,XOA REVERSEI /VXOD,XOA REVE1
 000051 //           RSEO"'
 000052 //STEPLIB  DD  DISP=SHR,DSN=&DTXLIB
 000053 //         DD  DISP=SHR,DSN=&XMLLIB
 000054 //*
```

Figure 6-70 Details DTXPROF options

Figure 6-71 shows the extended output for DTXPROF.

```
Individiual function breakdown
Function          Depth Iterations    Time      Map                      Type
-------------------------------------------------------------------------------
Validation
VALIDATE_CARD         0       1          8 reverse
 Mapping
BUILD_CARD            0       1         17 reverse
MAP_NAME              1       1          6 reverse            OutRecord Revers
TYPE_NAME             2       1          0 reverse            OutRecord Revers
EITHER                2       1          1 eachRecord         Line Fields Reve
EXIT                  3       1          1 eachRecord         Line Fields Reve
STRING_LITERAL        4       1          0 eachRecord         Line Fields Reve
STRING_LITERAL        3       1          0 eachRecord         Line Fields Reve
TYPE_NAME             2       1          0 eachRecord         Word1 Fields Rev
TYPE_NAME             2       1          0 eachRecord         Word2 Fields Rev
(….. . . . . )
TYPE_NAME             6       2          0 eachRecord         Length Fields R
 TYPE_NAME             5       1          0 eachRecord          Length Fields R
   Type per rule breakdown
       Time      Map                 Type
 ------------------------------------------------
 Validation
  Mapping
         25      reverse
          6      reverse             OutRecord Reverse
          2      eachRecord          Line Fields Reverse
          0      eachRecord          Word1 Fields Reverse
          0      eachRecord          Word2 Fields Reverse
          0      eachRecord          Word3 Fields Reverse
          6      eachRecord          Length Fields Reverse
          0      reverse             OutRecord Reverse
          0      eachRecord          Line Fields Reverse
(. . . . . .)
```

Figure 6-71 Extended outputs for DTXPROF

6.5 WebSphere Transformation Extender for z/OS: Application Programming

The IBM WebSphere Transformation Extender for Application Programming edition supports the following runtime environments:

- CICS
- IMS TM
- z/OS Software Development Kit
- UNIX System Services Software Development Kit (SDK)

With the IBM WebSphere Transformation Extender for Application Programming edition, WebSphere Transformation Extender Maps can be programmatically executed through Enterprise JavaBeans (EJB), Java, Remote Method Invocation (RMI), C/C++, COBOL, or PL/I programs.

6.5.1 The Platform APIs on z/OS

The Platform API is available as part of the SDK runtime execution environment that is included with the WebSphere Transformation Extender for z/OS. The SDK contains the Platform API runtime and example programs for C, COBOL, and PL/I languages. Throughout the Platform API documentation, references are made to the example files that are included in the WebSphere Transformation Extender installation.

More information: See the *Platform APIs* guide (`1020.pdf` file) at the following address:

`ftp://ftp.software.ibm.com/software/websphere/integration/wdatastagetx/`

WebSphere Transformation Extender can be tightly integrated with home grown applications and third-party software. The WebSphere Transformation Extender execution return code and status is returned to the calling client application. Transformation results can be returned to the calling client application or written to output by WebSphere Transformation Extender adapters.

Basic APIs: The following basic APIs can be used:

- *MercInitAPI()* establishes connections for WebSphere Transformation Extender adapters, including database, messaging, and utility.
- *MercExitAPI()* closes the adapter connections that are established by MercInitAPI() and performs necessary API cleanup.
- *RunMap()* executes the map or maps.

A typical usage of the API is to pass data through memory to the map. The passing of data is done by referencing data structures, such as EXITPARAM, and setting or getting values from it.

Passing input data to a map from a program

Use the Input Source Override - Echo execution command (**-IE**) when there is an application that has a pointer to data to use as a source to a map. The code fragment Example 6-15 shows one way to use the Input Source Override - Echo execution command (**-IE**) to pass that input data to the mymap.mmc map.

Example 6-15 Passing input data from a C program

```
EXITPARAM ExitParam;
LPBYTE lpBuf;
DWORD dwLen;
char szbuf[50];
int nlen;
. . .
/* dwLen is length of input data */ /* lpBuf is the buffer that contains the input
data */
/* -ie1 means "echo" in input for source number 1 */
/* in this example, we will use the size option with the -ie option to tell the API
the size of the input */
nlen = strlen(szbuf);
/* allocate storage space for lpDataToApp */
ExitParam.lpDataToApp = malloc(dwLen + 100);
ExitParam.dwSize = sizeof(EXITPARAM); ExitParam.lpv = NULL;
/* assign it to lpDataToApp */
memcpy(ExitParam.lpDataToApp, szbuf, nlen); memcpy(&ExitParam.lpDataToApp[nlen],
lpBuf, dwLen);
/* call the function to map it */
RunMap(&ExitParam);
/* lpDataFromApp contains the string representation of nReturn */
. . .
```

Returning data from a map to a program

Use the Output Target Override - Echo execution command (**-0E**) to return the data from a map target. Override an output by using the Output Target Override - Echo execution command (**-0E**). lpDataFromApp contains the data for the specified output card. The usage of **-0E** for more than one output concatenates the data for the outputs with no separator.

A cache mechanism is available when calling the same map more than once in a process. The instance of the map is kept until it is released.

Functions for a cache mechanism: A cache mechanism can be used with three functions:

- *InitializeRunMapInstance()* initializes a specific map file into memory for one or more subsequent executions.
- *RunMapUsesInstance()* sets or clears the EXITPARAM structure that is passed into the RunMap function for maps that have been initialized into memory by using the InitializeRunMapInstance function.
- *FreeRunMapInstance()* frees all memory that is associated with a map that has been initialized by using the InitializeRunMapInstance function.

WebSphere Transformation Extender Maps can be called synchronously (inline) or asynchronously. The same map developed to run in a Command Server or Launcher can be used inside an application by overriding the input and output source.

All other options and settings are available from the calling program:

- Turn on or off trace
- Turn on or off audit log
- Cards options

GetCardIOs() function: The GetCardIOs() function accesses card I/O settings. It obtains card and adapter settings for sources and targets for a particular map file.

Example files

The SINKMAP example map is the default map that is used by the following examples of the Platform API. The map, input, and JCL files are in the `DTX.SDTXSAMP` PDS that is included in the WebSphere Transformation Extender installation.

- Platform API C example

  The DTXTAPIC C example program demonstrates how to call all of the Platform API functions from a C program that is compiled as a DLL.

- Platform API COBOL example

  The DTXTCCOB COBOL example program demonstrates how to call all of the Platform API functions from a COBOL program that is compiled as a DLL.

- Platform API PL/I example

  The DTXTPLI PL/I example program demonstrates how to call all of the Platform API functions from a PL/I program that is compiled as a DLL.

Using the Platform API

To use the Platform API functions in C, COBOL, and PL/I programs, a specific procedure is required. The procedure assumes that a map or system *has been* previously created. It references the EXITPARAM API structure, which is described in the Platform API structure definitions (Example 6-16 on page 417).

Table 6-1 Platform API structure definitions

Component	Used as	Use
dwSize	Input	The size (in bytes) of the EXITPARAM structure to ensure correct version compatibility
dwFromLen	Output	The length (size in bytes) of lpDataFromApp
dwMapInstance1		Must be set to a unique number for each map that is executed
lpv		Not used and should be NULL
lpDataToApp	Input	The command line being passed in
lpDataFromApp	Output	The result based on command options
nReturn	Output	The return code based on the last processed map
szErrMsg	Output	String message based on nReturn

To use the Platform API:

1. From *within the* application, if the map uses an adapter such as a database, messaging, or utility, a call to the MercInitAPI() function is needed to establish adapter connections before calling any Platform API functions.
2. Initialize the EXITPARAM structure using a EXITPARAM structure, which will subsequently be used as an argument for the RunMap function.
3. Create a standard command line within the application.

4. If you called MercInitAPI() to establish connections for an adapter in step 1 , call MercExitAPI() after calling Platform API functions to close the adapter connections and perform necessary API cleanup tasks.

Note: Use the XPLINK(ON) option in the map command. Otherwise the "ABEND=S000 U4038 REASON=00000001" error message is displayed.

```
******************************** TOP OF DATA ********************************
CEE3555S A call was made from a NOXPLINK-compiled application to an XPLINK-compi
         and the XPLINK(ON) runtime option was not specified.
         From entry point main at statement 14 at compile unit offset +00000060
         21300EF8.
******************************** BOTTOM OF DATA *****************************
```

For COBOL, add the option in the WebSphere Transformation Extender command line, in the JCL or program:

```
'SINKMAP.MMC /VX15 ALLEMPS /XPLINK(ON)'
```

For C, use the XPLINK(ON) parameter at compile or link time.

Programming in COBOL

The following two copybooks are members in the DTX.SDTXSAMP PDS that is included in the installation of the WebSphere Transformation Extender feature:

- DTXRMCOB
- DTXMQCOB

When combined, these two copybooks make up the COBOL definition of the Platform API structures. The DTXRMCOB copybook contains the COBOL definition for both the ExitParam and Exeopts structures. The DTXMQCOB copybook contains the COBOL definition for the CARDINFO structure when using the WebSphere MQ adapter.

The DTXIAEXP file contains the DLL import statements for the Platform API functions. This file is required for the LE prelinker step of any COBOL or C/C++ compile job that wants to link in the support for the Platform API (Example 6-16).

Example 6-16 Simple COBOL program example

```
CBL RENT,DLL,NOEXPORTALL,MAP,LIST,LIB,PGMNAME(M),OPT(STD)
IDENTIFICATION DIVISION.
PROGRAM-ID. 'WTXCOB'.
DATE-WRITTEN. FEV  2007.
DATE-COMPILED.
-------------------------------------------------------------
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
```

```
*SOURCE-COMPUTER. IBM-370 WITH DEBUGGING MODE.
 SOURCE-COMPUTER. IBM-370.
*
 INPUT-OUTPUT SECTION.
 DATA DIVISION.
 FILE SECTION.
*
/
WORKING-STORAGE SECTION.
**********************
*
 77  W-PGM-EN-COURS              PIC X(08) VALUE 'CORBWTX'.
 77  MERI-RET-CODE               PIC S9(09) COMP VALUE ZERO.
*
LINKAGE SECTION.
copy DTXRMCOB.
*copy DTXMQCOB.
01  MAP-PARM.
     05  MAP-PARM-LG       PIC S9(4) COMP.
     05  MAP-PARM-TEXT     PIC X(100).
     05  FILLER  REDEFINES  MAP-PARM-TEXT.
         10  MAP-PARM-CHAR PIC X OCCURS 100.
*

PROCEDURE DIVISION USING MAP-PARM.
*********************************
*
 DEBPROG.
*
     DISPLAY 'DEBPROG'.
*
     IF MAP-PARM-LG = ZERO
           DISPLAY 'MISSING MAP PARAMETER'
           GO TO FIN.
     IF MAP-PARM-CHAR(MAP-PARM-LG + 1) = '/'
           MOVE SPACE TO MAP-PARM-CHAR(MAP-PARM-LG + 1).
     MOVE LOW-VALUE               TO  EXITPARM.
     MOVE LENGTH OF EXITPARM      TO  DWSIZE.
     DISPLAY 'MercInitAPI'.
     CALL 'MercInitAPI' RETURNING MERI-RET-CODE.
     IF MERI-RET-CODE NOT = ZERO
          DISPLAY 'ERROR MERCINITAPI ' MERI-RET-CODE
          GO TO FIN.

     DISPLAY 'RunMap'.
```

In this example, the PARMS options are taken from the JCL.

```
     SET LPDATATOAPP              TO  ADDRESS OF MAP-PARM-TEXT.
     CALL 'RunMap'  USING  BY  REFERENCE  EXITPARM.
     IF NRETURN > ZERO
          DISPLAY 'ERROR EXECUTION RUNMAP : ' NRETURN '/'
                                              SZERRMSG
     ELSE DISPLAY 'RUNMAP DONE'
     END-IF.
*
     DISPLAY 'MercExitAPI'.
     CALL 'MercExitAPI'.
*
 FIN.
     GOBACK.
*
```

Programming in C

The WebSphere Transformation Extender installation includes the DTXTAPIC example program, which is in the DTX.SDTXSAMP PDS. For details about how to call the Platform API from a C program, examine the following files that were installed as members in the DTX.SDTXSAMP PDS:

- DTXTARME is the readme file that contains instructions for running the DTXTAPIC program.
- DTXTRJCL is the sample JCL that demonstrates how to use the Platform API to compile and run the SINKMAP map example.
- DTXTAPIC is the sample C program that demonstrates how to use the Platform API to run the SINKMAP map.
- DTXRUNMH is the Platform API header file (.h).

Example 6-17 shows a C program with the Platform APIs.

Example 6-17 Simple C program with the Platform APIs

```
#include <stdio.h>
#include "runmerc.h"
int main(void) {
EXITPARAM ExitParam;
MercInitAPI();
memset(&ExitParam,0,sizeof(EXITPARAM));
ExitParam.lpDataToApp = (unsigned char *)
"sinkmap.mmc -ae -ts -p32:8\0";
ExitParam.dwSize = sizeof(EXITPARAM); ExitParam.lpv = NULL;
RunMap(&ExitParam);
printf("Map returned %d: %s\n", ExitParam.nReturn, ExitParam.szErrMsg);
```

In this example, the options are taken from the program.

```
if (ExitParam.dwfromLen) free (ExitParam.lpDataFromApp);
MercExitAPI();
return 0;
}
```

Programming in PL/I

The WebSphere Transformation Extender installation includes the DTXTPLI example program, which is in the DTX.SDTXSAMP PDS. The IBM Enterprise PL/I for z/OS compiler is used to compile the DTXTPLI example program.

For details about how to call the Platform API from a PL/I program, examine the following files that were installed as members in the DTX.SDTXSAMP PDS:

- DTXPRME is the readme file that contains instructions for running the DTXTPLI example.
- DTXPJRUN is the sample JCL that demonstrates how you can use the Platform API to run the SINKMAP map example.
- DTXTPLI is the sample PL/I program that demonstrates how you can use the Platform API to run the SINKMAP map example.
- The DTXPINC PL/I include file contains declarations of the EXITPARAM, EXEOPTS, and CARDINFO structures that are used in the SINKMAP map example. This include file also contains the declarations of the Platform API entry points.
- DTXPJCMP is the sample JCL that demonstrates how you can use the Platform API to compile and link the API program example.
- The PL/I include file describes the content for the EXITPARAM, EXEOPTS, and CARDINFO structures that are written in the C language.

6.5.2 The SDK on UNIX System Services

The available languages on UNIX System Services are C/C++ and Java. There are no differences with other UNIX platforms.

The WebSphere Transformation Extender Programming Interface is an object-oriented approach that enables applications to invoke Transformation Extender maps, thus removing reliance on command lines.

More information: See the following documents for more information APIs in terms of C APIs, Java, and EJB:

- C APIs: See the `1132.pdf` file in the IBM WebSphere Transformation Extender Online Library
- Java: See the `<install_dir>\docs\dtxpi\java\overview-summary.html` file.
- EJB: See the `1135.pdf` file in the `<install_dir>\docs\en\dtxpi\ejb` directory.

Using the Programming Interface

WebSphere Transformation Extender maps can be called as transformation services from WebSphere Application Server, WebSphere Process Server, and WebSphere Enterprise Service Bus (ESB). The WebSphere Transformation Extender Programming Interface enables applications to invoke WebSphere Transformation Extender maps and to develop adapters to use with the WebSphere Transformation Extender. Example 6-18 shows a Java program using the WebSphere Transformation Extender APIs.

Example 6-18 Java program using WebSphere Transformation Extender APIs on UNIX System Services

```
import com.ibm.websphere.dtx.dtxpi.MConstants;
import com.ibm.websphere.dtx.dtxpi.MException;
import com.ibm.websphere.dtx.dtxpi.MMap;

public class Example1
{
    public static void main(String[] args)
    {
        try
        {
            // Initialize the API
            MMap.initializeAPI(null);

            // Create a map
            MMap map = new MMap("test1.mmc");

            // Run the map
            map.run();

            // Check the return status
            int iRC = map.getIntegerProperty(MConstants.MPIP_OBJECT_ERROR_CODE, 0);
            String szMsg = map.getTextProperty(MConstants.MPIP_OBJECT_ERROR_MSG, 0);
            System.out.println("Map status: " + szMsg + " (" + iRC + ")");

            // Clean up
```

```
            map.unload();
            MMap.terminateAPI();
        }
        catch( MException e )
        {
            e.printStackTrace();
        }
    }
}
```

Using the C APIs

The C API enables applications to invoke maps and develop adapters to use with WebSphere Transformation Extender. Example 6-19 shows a C program.

Example 6-19 C program with UNIX System Services SDK

```
000012 £include "dtxpi.h"
 000013
 000014 £define CHECK_ERR(rc) if(MPIRC_FAILED(rc)) { printf("Last error is: %sç
 000015
 000016 int main()
 000017 {
 000018  const char   *szMsg;
 000019  int  iRC;
 000020  MPIRC           rc;
 000021  HMPIMAP  hMap;
 000022
 000023  rc = mpiInitAPI(NULL);
 000024  CHECK_ERR(rc);
 000025
 000026
 000027  rc = mpiMapLoadFile (&hMap, "test1.mmc");
 000028  CHECK_ERR(rc);
 000029 rc = mpiMapRun (hMap);
 000031  CHECK_ERR(rc);
 000032
 000033  rc = mpiPropertyGetText (hMap, MPIP_OBJECT_ERROR_MSG, 0, &szMsg, NULL);
 000034  CHECK_ERR(rc);
 000035  rc = mpiPropertyGetInteger (hMap, MPIP_OBJECT_ERROR_CODE, 0, &iRC);
 000036  CHECK_ERR(rc);
 000037  printf("Map status: %s (%d)çn", szMsg, iRC);
 000038
 000039  rc = mpiMapUnload (hMap);
 000040  CHECK_ERR(rc);
 000041
```

```
000042  rc = mpiTermAPI();
000043  CHECK_ERR(rc);
000044  return 0;}
```

6.6 WebSphere Transformation Extender for z/OS: CICS

WebSphere Transformation Extender includes the CICS Execution Option through which maps can run in CICS environments.

More information: See *Command Server* guide (`1010.pdf` file) and Chapter 4, “z/OS CICS” within that guide at the following address:

`ftp://ftp.software.ibm.com/software/websphere/integration/wdatastagetx/1010.pdf`

Loading and running a map by using the CICS Execution Option

To load a map by using the CICS Execution Option for z/OS CICS:

1. Build the map for z/OS. In the Design Studio, select **Map** → **Build for Specific Platform** → **IBM z/OS**.
2. Use file transfer to transfer the platform-specific map to z/OS CICS as BINARY.
3. Modify and execute the DTXCMJCL JCL to load the map into the z/OS CICS VSAM map data set.

Then you can invoke the CICS Execution Option in any of the following ways:

- From a clear terminal, enter the DTXI transaction and follow it with a command string that contains the name of at least one map and any execution commands that are required for the execution of that map. Command arguments must be separated by spaces. For example, to run the MYMAP map from a terminal with commands that specify the input and output card overrides, type the following command:

  ```
  DTXI MYMAP -I1 MYINPUT -O1 MYOUTPUT
  ```

- Start the DTXI transaction from a user program by passing the DTXI transaction name followed by the command string that contains the map name and any required execution commands in the FROM option of the EXEC CICS START command.

- Automatically initiate the DTXI transaction from a trigger transient data queue. The first record in the queue must contain the transaction name DTXI followed by the command string.
- Invoke the DTXI program by using an EXEC CICS LINK and pass the link control blocks. (Use the DTXSCTST sample program as a guide.) These control blocks define areas for the command string, optional data buffers, and a return code and message.

XPLINK and non-XPLINK: The CICS Execution Option takes advantage of the C/C++ Extra Performance Linkage (XPLINK) compiler optimization that was introduced in CICS Transaction Server for z/OS V 3.1. WebSphere Transformation Extender for z/OS includes `DTX.SDTXLOAD` and `DTX.SDTXLOD2` load libraries. The `DTX.SDTXLOAD` load library contains files that support the use of XPLINK. If you want to use XPLINK, you must add the `DTX.SDTXLOAD` load library to the DFHRPL concatenation.

6.7 WebSphere Transformation Extender for z/OS: IMS TM and IMS/DC

IMS on z/OS and MVS generally refers to a product that provides the following two functions:

- Transaction management (IMS TM or IMS/DC)
- Database management (IMS/DB)

The IMS/DC Execution Option refers to the transaction management function. This option provides a means to use the Platform APIs for z/OS and high-level language programs to programmatically call the WebSphere Transformation Extender execution environment. The high-level languages that are supported on z/OS are C/C++ and COBOL.

More information: For documentation, see the *IMS/DC Execution Option* guide (`1103.pdf` file) at the following address:

`ftp://ftp.software.ibm.com/software/websphere/integration/wdatastagetx/`

See also the *z/OS Configuration* guide (`1145.pdf` file) in the IBM WebSphere Transformation Extender Online Library.

To call Platform API functions as a DLL, set up the parameters to pass to the Platform API.

The RunMap function takes only one parameter, which is the ExitParam structure address. When using the RunMap function, the pointer to the map file name and the pointer to the DataFromApp are in the ExitParam structure.

Setting the address of the map file name requires that the map file name be passed through the JCL PARM statement (linkage section). Alternatively, it requires that a subprogram be created that takes the address of the map file name as one parameter and passes the address back, in the form of a pointer, in another parameter. This subprogram deviation of the CALLRMAP function was thought to simplify its use.

A cache mechanism for maps and the .mdq preloader can be used. This is an optional user exit routine that provides a way to preload WebSphere Transformation Extender map and .mdq files into memory. All subsequent access to these maps and .mdq files during map execution are from memory. Therefore, they do not require any file input or output processing from these preloaded map and .mdq files.

6.8 WebSphere Transformation Extender for z/OS: Command Server, Launcher, and SDK for UNIX System Services

WebSphere Transformation Extender Command Server is also available for the UNIX System Services environment. Maps are launched from a command line with standard options for UNIX systems.

WebSphere Transformation Extender Launcher runs in the UNIX System Services environment. The Launcher is an event-driven, multi-threaded, multi-process transformation engine. The WebSphere Transformation Extender Launcher process is always active and waiting for events to occur. The Launcher manages events such as file creation (data sets), modification, MQ file queue triggering, and so on.

Triggers include the following supported events among others:

- Time events: Single instance or recurring
- Source events: Files (native and hierarchical), WebSphere MQ, Oracle
- Compound events
  - Multiple sources
  - Source and time

Both Command Server and Launcher support the hierarchical and native file systems (sequential files, PDS members, and PDS extended (PDSE) members).

Note: Using a DB2 data source as an input event trigger for the Launcher is not supported.

With the WebSphere Transformation Extender Development Kit (SDK) for UNIX System Services, you can run the development kit on z/OS operating systems that run on the mainframe in a UNIX environment.

The WebSphere Transformation Extender SDK for UNIX System Services has the following characteristics:

- The application is 31 bits.
- DSTXPI consists of six APIs for C, C++, Java, and RMI.
- Java-based resource adapters and API interfaces are available.

Command Server, Launcher, and APIs on UNIX System Services can use the same maps as in z/OS batch mode.

Default extension: The .mvs file extension can be changed to the .mmc extension, because it is the default extension that is used by Command Server and Launcher on UNIX System Services.

UNIX System Services shell: UNIX System Services provides a UNIX shell on the z/OS environment. This shell includes a hierarchical file system that is familiar to UNIX users. The default encoding is EBCDIC. It has the capacity to communicate with the primary MVS subsystems (DB2, IMS, CICS, and WebSphere MQ). See Figure 6-72.

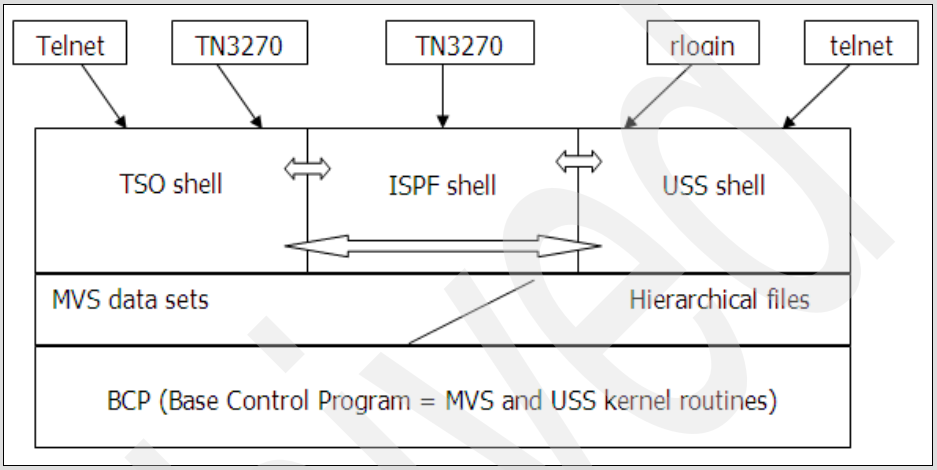

Figure 6-72 UNIX System Services subsystem

6.8.1 Post installation in UNIX System Services

To use WebSphere Transformation Extender, you must create a user. That is an executable environment.

More information: See *z/OS Configuration* (`1145.pdf` file) in the IBM WebSphere Transformation Extender Online Library.

To create a runtime environment, adapt and launch the DTXINST JCL in the DTX.DSTXSAMP PDS as shown in Figure 6-73. Launching the JCL creates all directories for the execution environment and a script for setting all necessary variables for WebSphere Transformation Extender before using it.

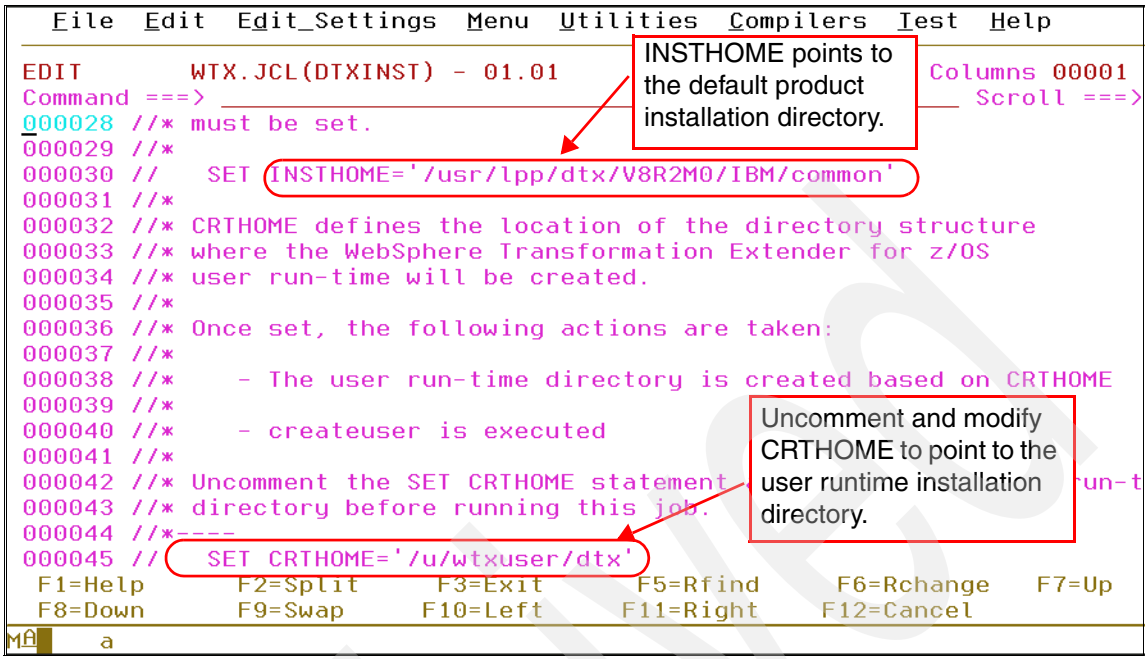

Figure 6-73 JCL to create a runtime environment on UNIX System Services

Important: You must have read access to the INSTHOME directory files or change to a superuser.

Check and modify, if necessary, the setup script to set the environment variables PATH, DTX_TMP_DIR, and DTX_HOME_DIR and the appropriate library path variable for your platform:

1. Set the location for the temporary files (DTX_TMP_DIR).

   During map execution, the Command Server creates temporary files for resource handling purposes and for retaining debug information. These temporary files use the `/tmp` directory as the default. However, you can specify the directory where you want these files to be kept by setting the DTX_TMP_DIR environment variable, as shown in Figure 6-74.

```
DTX_TMP_DIR=install_dir/tmp
  export DTX_TMP_DIR
```

Figure 6-74 Temporary files location

Note: If you choose to allow multiple groups to access the Command Server engine, the directory specified for DTX_TMP_DIR requires permission 777 to provide permission to the user, the group, and all others.

To grant permission 777, run the **chmod** command as follows:

```
chmod 777 $DTX_TMP_DIR
```

In addition, for multiple users or groups, define a DTX_TMP_DIR instead of letting it default to the /tmp directory.

2. Set the shared libraries environment variables:

   a. Set the XMLTOOLKIT path.

      Usually the XML Toolkit is in the /usr/lpp/ixm/IBM/xml4c-5_6/lib directory. In the setup file, add it to the LIBPATH variable as follows:

      ```
      export LIBPATH
      /u/wtxuser/wtx/libs:/usr/lpp/ixm/IBM/xml4c-5_6/lib:$LIBPATH
      ```

      Note: When the XML Toolkit is not accessible for WebSphere Transformation Extender, the following error message is displayed:

      ```
      CEE3501S The module IXMI33XC was not found.
      The trace back information could not be determined. Ý1~ +
      Done(137) dstx maps/reverse.mvs
      ```

   b. Set the Java path as shown in the following example:

      ```
      export JAVAHOME=/usr/lpp/java/J5.0
      ```

   c. Export the _CEE_RUNOPTS environment variable.

      Consider the situation where you are using IBM version of Java 2 Runtime Environment (JRE™) on a z/OS operating system and you want to use the WebSphere Transformation Extender in your UNIX System Services runtime environment. In this case, you must add XPLINK to the _CEE_RUNOPTS UNIX System Services environment variable as follows:

      ```
      export _CEE_RUNOPTS=$_CEE_RUNOPTS,"XPLINK(ON)"
      ```

Example 6-20 shows the setup shell.

Example 6-20 Setup shell

```
#!/bin/sh
# Licensed Materials - Property of IBM;
# (c) Copyright IBM Corp. 2006, 2007
# 5655-R95
```

```
# WebSphere Transformation Extender for z/OS setup script
DTX_HOME_DIR=/u/wtxuser/dtx
export JAVAHOME=/usr/lpp/java/J5.0
export LIBPATH=/usr/lpp/ixm/IBM/xml4c-5_6/lib:$LIBPATH
#export DISPLAY=SC59:0
export _BPX_SHAREAS=YES
export _CEE_DMPTARG=/u/wtxuser/dtx/dump
export _IXM_FORCE_CONVERSION=USE_ICU_SRC_OFFS
. /u/wtxuser/dtx/bin/envsetup /u/wtxuser/dtx
```

After you log on to your system, execute the setup program in the WebSphere Transformation Extender installation directory, before you execute a map or system file (using the Launcher). This sets the required environment variables for this session only.

Note: Modify the .profile script to set the environment variables for all sessions.

Enter the following command, where *install_dir* represents the directory in which you have installed your WebSphere Transformation Extender products:

```
. /install_dir/setup
```

Syntax: There must be a space between the initial period (.) and the command path.

Finally check all maps to be sure that they comply with the default (NATIVE) code page of EBCDIC. This is important when using a RUN function to call another map in a rule or when using an adapter.

Note: If you are running maps on the UNIX System Services execution environment, the native character set is EBCDIC, which is invalid for the Java adapter. To keep the character set unchanged, use the NATIVE keyword.

Change the following rule:

```
GET("JAVA","-T",PACKAGE(GROUP:RECORD))
```

Change the rule as follows to include the Native keyword:

```
GET("JAVA","-T",CPACKAGE(GROUP:RECORD,"NATIVE"))
```

6.8.2 Running a map

You must compile the map for z/OS (a .mvs extension) and then deploy it with a .mmc extension name to use the defaults. If the map was copied into the maps subdirectory of $CRTHOME, start WebSphere Transformation Extender by using the **dstx** equivalent **dtxcmd** command as shown in Example 6-21.

Example 6-21 Running a map command

```
./bin/dtxcmdsv maps/reverse.mmc -IF1 /u/wtxuser/dtx/data/reversei.txt
-OF1 /u/wtxuser/dtx/data/reverseo.txt
```

In Example 6-21, the /u/wtxuser/dtx path is the CRTHOME directory where the user has been created.

Note: When overriding files names with the **-IFn** or **-OFn** option in command line, use an absolute path to name files, not relative names.

Test with a native input file, a data set, or PDS member, with an implicit high-level qualifier (your logon user ID). Figure 6-75 shows an example of using a sequential data set.

```
./bin/dtxcmdsv maps/reverse -IF1 //'WTX.DATA.REVERSEI'
```

Figure 6-75 Sequential dataset example

Figure 6-76 shows an example of using a PDF member with a native input file.

```
./bin/dtxcmdsv maps/reverse -IF1 //'WTX.(DTXRTEXT)'
```

Figure 6-76 PDS member example with a native input file

Note: When input/output files are not mentioned in the command line, they must be in the directory that is defined in the card. If just the name of a file is given in the card, the files are in the same directory as the map file.

Test with a native input file, but with an explicit high-level qualifier. Place both double quotation marks and single quotation marks around the name, as shown in Figure 6-77.

```
./bin/dtxcmdsv maps/MINSCT2.mmc -IF1 //"'XXXX.WTX.DATA.F2OP180'"
-OF1 /u/wtxuser/dtx/data/out.txt
```

Figure 6-77 Using a native file

Test with a map in a native file (that is a member of a PDS) and a local HFS file (Figure 6-78).

```
dstx //"'WTX.MAPS(REVERSE)'" -IF1 /u/wtxuser/dtx/data/reversei.txt
```

Figure 6-78 Using a native file with an HFS file

The iconv utility: To look at XML files with encoding UTF-8 with the standard editor OEDIT, use the **iconv** utility. For example, you can convert a file from UTF-8 to EBCDIC as follows:

```
iconv -f UTF8 -t IBM-1047 ficin > ficout
```

You can use the following command for a native file:

```
iconv -f UTF8 -t IBM-1047 //'WTX.DATA.ORDERSX ' > InEbcdic.xml
```

6.9 WebSphere Transformation Extender on Linux on System z

All WebSphere Transformation Extender components are available on Linux on System z. WebSphere Transformation Extender version 8.2 offers 64-bit operating system tolerance support for selected Linux on System z environments:

- SUSE Linux Enterprise Server (SLES): Currently SLES9 SP4 and SLES10 SP1
- Red Hat Enterprise Linux (RHEL): Currently RHEL4 U7 and RHEL5

When connected to Linux on System z, the installation and usage of WebSphere Transformation Extender are the same for the user.

What the System z platform brings to Linux:

- The most reliable hardware platform available
- Designed to support mixed workloads
- Complete workload isolation
- High speed inter-server connectivity
- Hundreds of Linux virtual servers
- Centralized Linux systems are easier to manage
- Shared application workloads for better utilization
- Dynamic resource allocation, based on demand
- Business Integration: speed, security and data management
- Linux on System z running side-by-side with z/OS, z/VM® VSE/ESA, provides best integration capabilities for applications and data
- Vastly efficient network, almost at memory speed
- Best performance, simplified management, more secure
- Rapid access to enterprise data and applications

Installation of the product is standard for a Linux product:

1. Transfer the tar installation file as shown in Figure 6-79 by using FTP.

```
ftp> bin
200 Switching to Binary mode.
ftp> put CdeServer_zLinux_C1I66ML.tar /tmp/wtxCS8203.tar
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 File receive OK.
ftp: 119377920 bytes sent in 1158.45Seconds 103.05Kbytes/sec.
```

Figure 6-79 Uploading the installation file

2. Untar the file as shown in Figure 6-80.

```
tar -xvf wtxCS8203.tar
```

Figure 6-80 Untarring the installation file

Figure 6-81 shows the result.

```
wtxuser@srvswg1:/tmp/wtx> ls
DTXINST license LICENSE.TXT messages README.TXT SELECT_COMP
WSDTXCS26.ZLINUX WSDTXCS.ZLINUX
```

Figure 6-81 Installation files

The product is installed in the `opt/IBM` directory, so that the superuser or correct permissions to create a directory there are required.

3. Start the installation as shown in Figure 6-82.

```
alain@srvswg1:/tmp/wtx> ./DTXINST
```

Figure 6-82 Starting the installation

During the installation, a list of available components is displayed as shown in Figure 6-83.

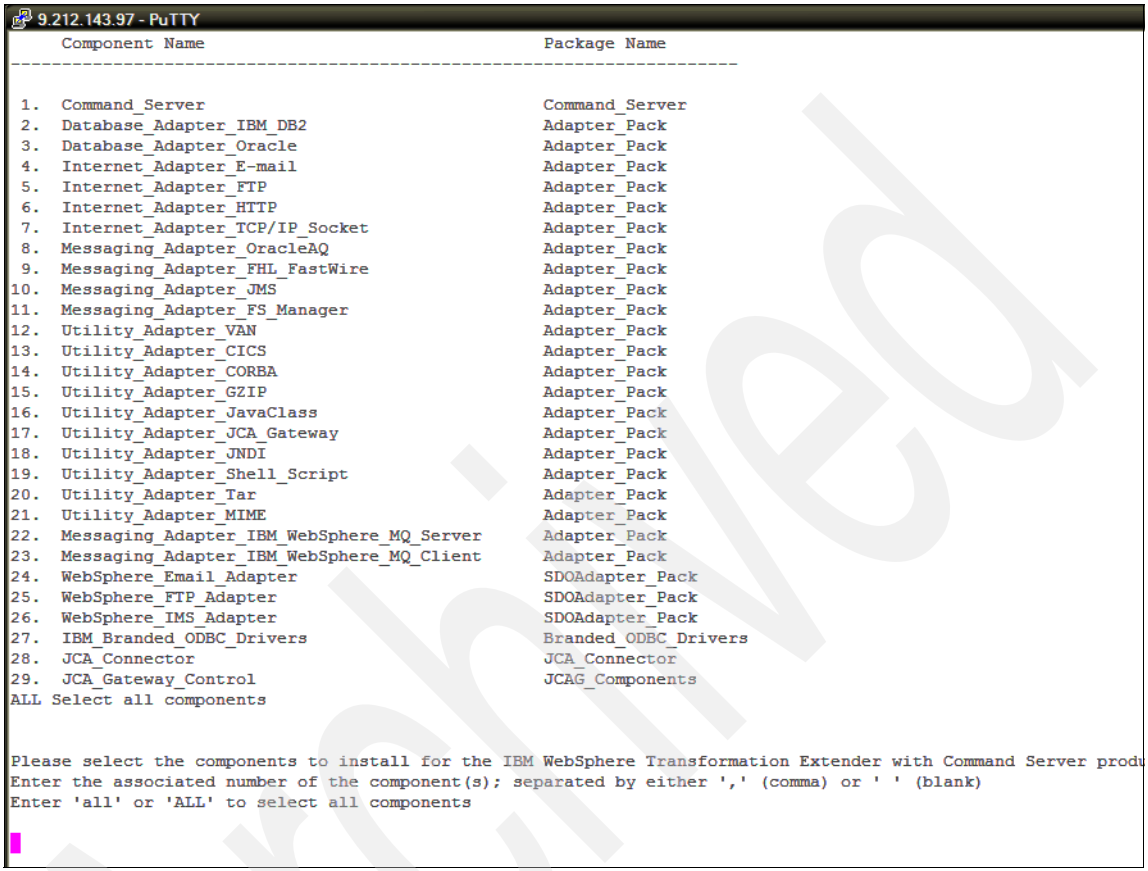

Figure 6-83 WebSphere Transformation Extender Components available on Linux on System z

Before using WebSphere Transformation Extender, set the execution environment variables by executing the setup script in the install directory. Figure 6-84 shows the command to set the execution environment variables.

```
opt/IBM/WTX/v8203/examples/general/map/sinkmap #. /opt/IBM/WTX/v8203/setup
```

Figure 6-84 Setup script for the runtime environment setting

Figure 6-85 shows the complete setup script.

```
 DTX_HOME_DIR (IBM WebSphere Transformation Extender Home Directory):
     /opt/IBM/WTX/v8203
   DTX_TMP_DIR (IBM WebSphere Transformation Extender Temp Directory):
     /opt/IBM/WTX/v8203/tmp
   LD_LIBRARY_PATH:
/opt/IBM/WTX/v8203/libs:/opt/IBM/WTX/v8203/java/bin:/opt/IBM/WTX/v8203/java/bin/
classic
   PATH::
/opt/IBM/WTX/v8203/bin:/opt/IBM/WTX/v8203/java/bin:/opt/IBM/WTX/v8202/bin:/opt/IBM/
WTX/v8202/java/bin:/home/alain/bin:/usr/local/bin:/usr/bin:/sbin:/usr/X11R6/bin:/
usr/sbin:/bin:/usr/games:/opt/gnome/bin:/usr/lib/mit/bin:/usr/lib/mit/sbin
```

Figure 6-85 Setup script

A sample map is provided in the <install_dir>examples/general/map/sinkmap directory. Figure 6-86 shows an example of launching the sinkmap map.

```
srvswg1:/opt/IBM/WTX/v8203/examples/general/map/sinkmap # dstx sinkmap
IBM WebSphere Transformation Extender(TM) Linux(i686) Command Server - version
8.2.0.3(58)_64bit
© Copyright 2008 IBM Corporation. All rights reserved.
Map file: /opt/IBM/WTX/v8203/examples/general/map/sinkmap/sinkmap.mmc
In # 2:       88  Out # 3:       22  Time:    0:00
Map completed (0):  Map completed successfully (0.033 seconds)
```

Figure 6-86 Launching the sinkmap map

WebSphere Transformation Extender with Launcher, WebSphere Transformation Extender for Integration Servers, and WebSphere Transformation Extender for Application Programming components can be installed the same way.

After installation of WebSphere Transformation Extender for Integration Servers, the WebSphere Transformation Extender node for WebSphere Message Broker is ready to be deployed as shown in Figure 6-87.

```
srvswg1:/opt/IBM/WTX/v8203 # ls wmqi
dtxwmqi_64.sh  dtxwmqi.jar  dtxwmqi.lil  messages
```

Figure 6-87 WebSphere Transformation Extender node for WebSphere Message Broker install directory

6.10 WebSphere Transformation Extender for z/OS: Integration servers

The following WebSphere Transformation Extender for z/OS products are packaged as WebSphere Transformation Extender for Integration Servers:

- WebSphere Transformation Extender for Message Broker
- WebSphere Transformation Extender for WebSphere Process Server and WebSphere ESB

The environment used for z/OS is UNIX System Services.

More information: For specific documentation, see the *z/OS Configuration* guide (`1145.pdf` file) in the IBM WebSphere Transformation Extender Online Library.

6.10.1 WebSphere Transformation Extender and WebSphere Message Broker on z/OS

You must complete the post installation in UNIX System Services. You must also ensure that the WebSphere Transformation Extender installation directory *and* user directory are accessible by the Message Broker. Give special attention to the temporary files directory because the files must be in read/write (R/W) mode.

Note: You can use the following additional features with the WebSphere Transformation Extender version 8.2.0.3 for WebSphere Message Broker, compared to the previous version:

- Resource Registry
- Map caching
- Map settings
- Multiple inputs
- Multiple outputs
- Additional supported message domains
- Wildcards
- Optional: Automatic deployment of maps when you deploy message flows

Configuring WebSphere Transformation Extender and WebSphere Message Broker

As a result of post installation in UNIX System Services, which is running the DTXINST JCL, the process creates the `dtxmqsi.sh` shell script, `/wtx_dir/mqsi/nn/dtxmqsi.sh`. The script is created in the IBM WebSphere

Transformation Extender for Message Broker z/OS runtime environment that it defines. In the script, *nn* is the release number of the WebSphere Transformation Extender Message Broker plug-in. For WebSphere Message Broker, the release number is 82. Therefore, the script is `wtx_dir/mqsi/82/dtxmqsi.sh`.

The wtx_dir location refers to the z/OS runtime environment where the DTXINST JCL job deployed the installed IBM WebSphere Transformation Extender for Integration Servers z/OS feature.

To use the WebSphere Transformation Extender node, the Message Broker environment file (ENVFILE) must be updated with the exported environment variables that are required by WebSphere Transformation Extender.

With WebSphere Transformation Extender version 8.2.0.3, the `dtxwmqi.ini` file has been removed. The trace, log, and resource registry settings are no longer available in the `dtxwmqi.ini` file. WebSphere Transformation Extender for Message Broker uses the WebSphere Message Broker interfaces instead.

Note: In previous versions of WebSphere Transformation Extender node (8.1), the `dtxwmqi.ini` file was used to set the log and parser paths. The default files were Windows like notations and made the broker fail when loading the node.

To prevent this failure, change the `/wmqi/dtxwmqi.ini` file as follows:

1. Change `FilePath="c:\n.log"` to `FilePath="<wtx_install_dir>/n.log"`.
2. Change `FilePath="c:\p.log"` to `FilePath="< wtx_install_dir > "`.

Do not leave this file with the defaults. Be sure to give read/write access to broker for these files.

To update the ENVFILE:

1. Add the following (dot command) line at the end of the BIPPROF member of your component data set (<HLQ>.BROKER.JCL):

   ```
   . <wtx_directory>wmqi/82/dtxwmqi.sh
   ```

2. Submit the BIPGEN JCL member from your component data set (<HLQ>.BROKER.JCL). The BIPGEN JCL invokes the shell script that is specified in the command line that you added to the BIPBPROF profile member, which sets the required environment variables.

Attention: Ensure that the permission bits are correctly set on the WebSphere Transformation Extender files to allow execution by Message Broker.

3. Stop the broker.
4. Run the WebSphere Message Broker profile that is created by BIPGEN.
5. Start the broker.

   In the log, ensure that there is no error and that the WebSphere Transformation Extender node has been referenced.

Using the WebSphere Transformation Extender Map node with a source map

During the WebSphere Message Broker process to build the flow, a WebSphere Transformation Extender node is used through the source map property. In this case, nothing special must be done for the z/OS broker. The maps are compiled for z/OS systems and deployed automatically. You create and build the broker archive (BAR) file and then deploy it to the broker.

Note: In this case, there is no need to deploy the map, because it is part of the Message Broker deployment process.

Using the WebSphere Transformation Extender Map node with a precompiled map

The WebSphere Transformation Extender node refers to an already built map. On the **Basic** tab of the WebSphere Transformation Extender Map node, the precompiled map option is selected.

The BAR file is manually created, built, and then deployed to the broker. The maps (.mmc files) are not in a .mar file and, therefore, not in the archive .bar file used for deployment. The compiled map (.mmc) file must be deployed manually to the broker's file system.

To upload the maps to a z/OS broker, use an FTP tool (or PCOM) in binary mode and send the .mvs compiled map.

Note: In this case, because the maps are not in the .mar file, they are not in the .bar file that is used to deploy resources to a broker.

6.10.2 WebSphere Transformation Extender and WebSphere Process Server or WebSphere ESB

With WebSphere Transformation Extender for WebSphere Process Server, the WebSphere Process Server or WebSphere ESB can use default WebSphere Transformation Extender bindings. These bindings are ready to deploy WebSphere Transformation Extender maps in mediation flows and business processes.

WebSphere Transformation Extender can also be invoked in a Service Component Architecture (SCA) component, using a Java snippet, and WebSphere Transformation Extender APIs. The problem is then how to define parameters for invocation of WebSphere Transformation Extender from WebSphere Process Server/WebSphere ESB.

WebSphere variables to define in WebSphere Application Server (defined at server level)

The WAS_SERVER_ONLY_server_region_classpath variable must include a path for WebSphere Transformation Extender classes, for example the `/Zxxx//u/wtxuser/dtx/libs/dtxpi.jar` path.

The WAS_SERVER_ONLY_server_region_libpath variable must include the path for WebSphere Transformation Extender and XML parser libraries, for example, the `/Zxxx//u/wtxuser/dtx/libs:/Zxxx/usr/lpp/ixm/IBM/xml4c-5_5/lib` path.

The following variables remain to be defined in WebSphere Application Server, with the default suggested values shown:

- Variable `DTX_HOME_DIR`; for example `/u/wtxuser/dtx`
- Variable `DTX_TMP_DIR`; for example `/tmp`
- Variable `_BPX_SHAREAS=NO`
- Variable `_CEE_DMPTARG=/tmp`
- Variable `_IXM_FORCE_CONVERSION=USE_ICU_SRC_OFFS`

Location of WebSphere Transformation Extender maps

When WebSphere Transformation Extender is called by Java code in a snippet, WebSphere Transformation Extender maps can be either of the following locations:

- The default WebSphere Application Server directory, which is named with a hardcoded path
- A specific directory defined by a variable, for example where the `wtx.ext.dir` environment variable is defined for the `/Zxxx/etc/wascfg/epcell/epnode1/AppServer/profiles/default/wtxlink/maps` map location.

In the case of WebSphere Transformation Extender used in bindings, the map name is either predefined by the template given in the binding and the name of business objects or is hardcoded. The maps are generally named without a file extension, so that it is possible to deploy the flows on any platform. In all cases, with WebSphere ESB or WebSphere Process Server, maps must be deployed manually. You must copy or transfer (using FTP) the compiled map with the file extension that corresponds to the platform.

On UNIX System Services, the maps use the .mmc default file extension, which means that the map name must be changed from the .mvs extension to the .mmc extension.

Note: Proper security must be set up so that it is possible to access the maps and write trace and audit log if active.

7

Troubleshooting

In this chapter, we provide information about the troubleshooting methods and capabilities of WebSphere Transformation Extender.

7.1 Approach to troubleshooting

Troubleshooting can be done in numerous ways. As a starting point, you must first determine where the problem is located. Namely, you must determine whether it is an issue related to proper input or an issue related to the output (Figure 7-1).

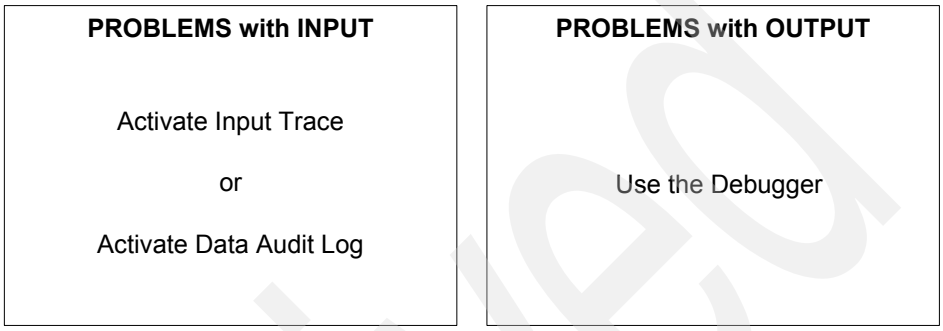

Figure 7-1 Troubleshooting methods depend on where the problem is located

Based on field experience, you might consider activating an input trace or data audit log for an input issue, and using the debugger for an output issue. It is important to understand that the run result does not provide sufficient information, because a map can run successfully as an execution process but not produce the right transformation. Beware of a return code of 0, which means that the map was able to be processed.

7.2 Validation

You must understand some validation basics when trying to troubleshoot map execution for WebSphere Transformation Extender. Validation is the first step of map execution. The data for the input cards is compared against the definition of the objects in the corresponding input type trees.

After all input cards are validated *and* free of errors, the map transformation starts. It generates output in temporary files (default) or memory. Then it connects to applications to generate physical outputs. This is the default, because we can force a card to generate physical outputs as soon as it has been built for that card. Any input card that fails validation generally stops further processing of the map.

7.2.1 Data validation

The validation process starts in a sequential order, at the highest object level of the first component, and ends with validation of the card level, which is level 0. For example in Figure 7-2, running a map starts the validation process of each input card, in sequential order.

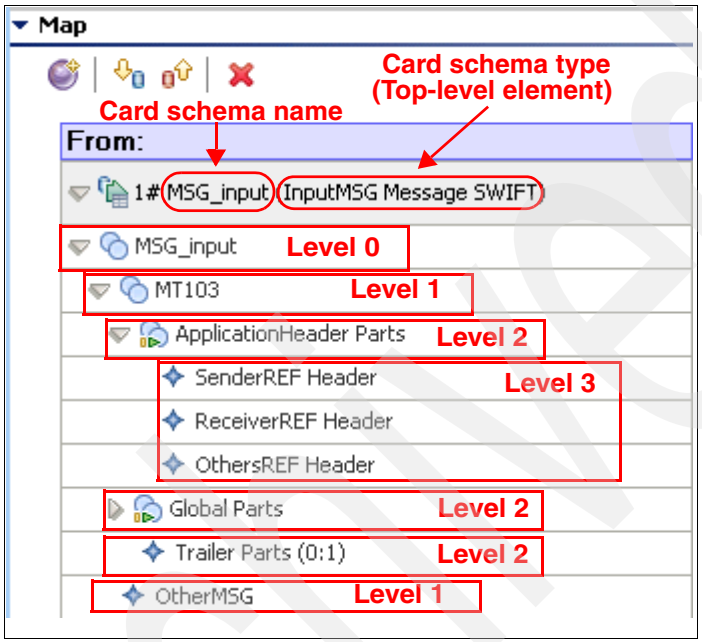

Figure 7-2 Validation process and type levels from Map editor view

Validation starts at offset 0 of the data stream and validates by using the first component of Level 3 (SenderREF Header). It then goes to the next offset and uses the second component of Level 3 (ReceiverREF Header). It moves to the next offset and uses the third component of Level 3 (OthersREF Header). Each object has a default (1:1) range. If this is not the case, the validation process validates each occurrence. Because the first (and in our case unique) occurrence of Level 3 objects is valid, the first occurrence of the Level 2 object is validated next.

When doing item validation, an item is valid if it has the following characteristics:

- The type is valid.
- The item range is valid.
- The component rule (if any) evaluates to TRUE.

When doing group validation, a group is valid only if all of its components are valid. A component is valid if it has the following characteristics:

- The type is valid.
- The component range is valid.
- The component rule (if any) evaluates to TRUE.

For card validation, if all objects in a card data object are found, then the card is valid. A trace indicates that each input is valid (Figure 7-3).

```
(Level 1: Offset 80112, len 6648, comp 2 of 2, #13, DI 00000482:)
Data at offset 80112 ('  HDR           ') was found to be of TYPE
   X'002C' (Message Record CopyBook).

(Level 0: Offset 0, len 86760, comp 1 of 0, #1, DI 00000483:)
Data at offset 0 ('     TYPE2       ') was found to be of TYPE
   X'0003' (DOC Record CopyBook).

   INPUT 1 was valid.

   End of Validation messages for INPUT CARD 1.
```

Figure 7-3 Trace of a valid input card

The flow chart in Figure 7-4 shows how input validation works.

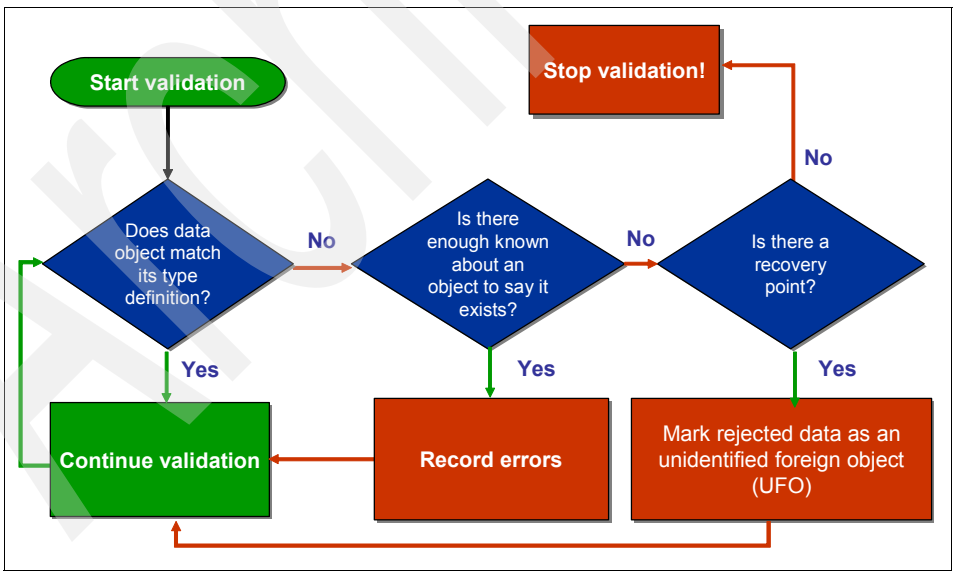

Figure 7-4 How input validation works

Invalid data

When a data object does not match the definition of the type to which it presumably belongs, it might be invalid. A data object might be invalid for the following possible reasons:

- The item value is not in a restriction list.
- The item failed the presentation test.
- The item is the wrong size.
- The data object has the wrong delimiter, initiator, or terminator.
- The data object failed the component rule test.
- The group is missing a required component.
- The group contains one or more invalid components.

Map return codes

When a map completes, a return code is returned that indicates the success, with or without some warnings, or failure of the map.

The following return codes are some of the common ones:

- One or more inputs was invalid (return code 8)

  This return code occurs when the entire input object is invalid. Turn on the input trace, run the map again, and read the trace file.

- Input valid but unknown data found (return code 21)

  This return code occurs when enough of the input data to conform to the card definition is recognized, but there is more data at the end of the input stream. Turn on the input trace, run the map again, and read the trace file.

- Input type contains errors (return code 28)

  This return code occurs when mapping data of a component that has a RESTART assigned to it, and at least one instance of the component in the input data is invalid.

Review the map execution and look for messages to help correct any map problems.

Command Server window

The Command Server window (Figure 7-5) is displayed from the Map Editor when you run a map or from command line execution. It shows the progress of validation and processing of data, which can help identify a card with problems. It indicates the result when the map completes.

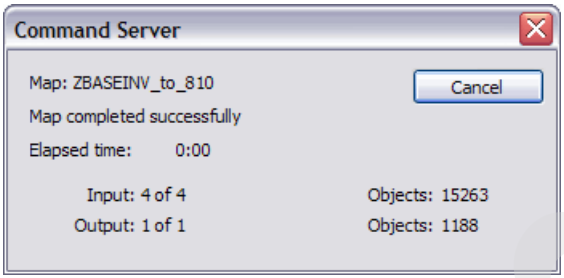

Figure 7-5 Command Server window with successful map execution

If the map does not complete successfully, you might see a validation error message. This type of error message usually indicates a discrepancy between the input data and the input type tree. It might show one of the following messages (Figure 7-6):

- One or more inputs was invalid
- Input valid but unknown data found
- Input type contains errors

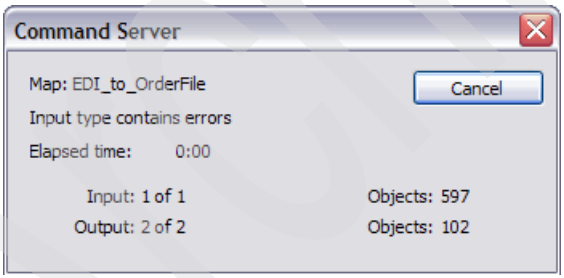

Figure 7-6 Command Server window with run error #28

Map audit log

For troubleshooting, enable the map audit log. Use the MapAudit settings in the Map Editor or the Integration Flow Designer (Figure 7-7).

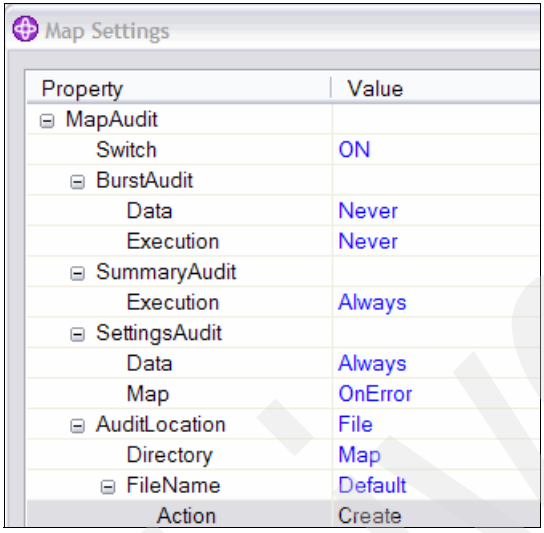

Figure 7-7 MapAudit settings

Use the Audit execution command (**-AE**) with the Command Server or the RUN( ) function. The audit log is an XML format file that shows selectable segments of information for monitoring map execution as a transformation process. It returns relevant information depending on activated segments:

- *Data Audit* shows the data itself.
- *Execution Summary* shows the most important segment, including all information about the way the map was executed (return code and message, and so on).
- *Data settings* show information about each input and output card setting.
- *Map settings* show information about the map settings.

You can activate each segment Always (equivalent of ON), Never (equivalent of OFF), OnError (in case of an error), or OnWarningorError (in case of a warning or error).

Switching the audit log on or off: You can use the global switch in the map settings to switch ON or OFF the audit log. However, switching the audit log to ON is not enough to produce an audit file. You must still activate one of the audit segments.

An audit log can be used in any environment, including a production environment. It is possible to generate audit log segments based on criteria (OnError, on error or warning), and generate a unique file per execution. It helps monitoring at the transaction level.

In the Map Editor, you configure the map audit log by selecting **Window** → **Show View** → **Audit**. You can then review the log to find details about the map execution (Figure 7-8).

```
<MapAudit StartTime="11:10:32 October 6, 2008">

<Platform> Platform API for Windows - Version 8.2(52)</Platform>

<ExecutionSummary MapStatus="Warning" mapreturn="28" ElapsedSec="0.0973" BurstRestartCount="0">
   <Message>Input type contains errors</Message>
   <CommandLine>'C:\Documents and Settings\Administrator\IBM\wtx\workspace\X12\Maps\EDI_to_OrderFile.mmc'</Comm
   <ObjectsFound>597</ObjectsFound>
   <ObjectsBuilt>102</ObjectsBuilt>

<SourceReport   card="1"   adapter="File"   bytes="4058"   adapterreturn="0">
   <Message>Data read successfully</Message>
   <Settings>C:\Documents and Settings\Administrator\IBM\wtx\workspace\X12\Maps\..\Data\Bad EDI_IN.txt</Setting
   <TimeStamp>13:32:12 July 9, 2007</TimeStamp>
</SourceReport>

<TargetReport   card="1"   adapter="File"   bytes="637"   adapterreturn="0">
   <Message>Data written successfully</Message>
   <Settings>C:\Documents and Settings\Administrator\IBM\wtx\workspace\X12\Maps\850sOut.txt</Settings>
   <TimeStamp>11:10:32 October 6, 2008</TimeStamp>
</TargetReport>

<TargetReport   card="2"   adapter="File"   bytes="2963"   adapterreturn="0">
   <Message>Data written successfully</Message>
   <Settings>C:\Documents and Settings\Administrator\IBM\wtx\workspace\X12\Maps\EDI with Errors.txt</Settings>
   <TimeStamp>11:10:32 October 6, 2008</TimeStamp>
</TargetReport>
```

Figure 7-8 Audit view of map audit log

Management Console

The Management Console is used to display runtime statistics of the Launcher. The Management Console can display results on the History tab and then on the Map and Function Failure subtabs of the History tab (Figure 7-9). See “The Launcher Management Console setting” on page 112 for further details about Management Console.

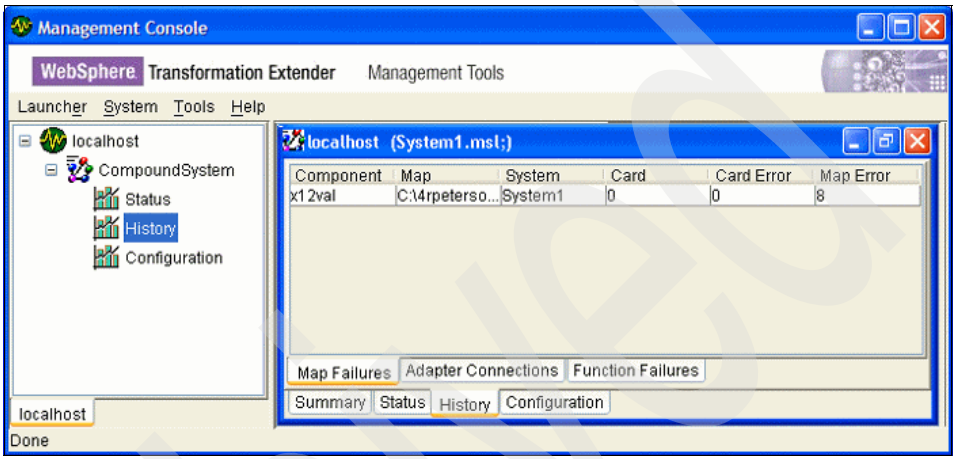

Figure 7-9 Management Console

Map input trace file

The map input trace file records the progress during the validation of the input data. It provides a step-by-step account of the data objects that are found. It also provides the reasons for any validation errors, if any.

Trace activation: Activating trace *significantly* affects performance. It generates at least three lines of information per object. Use trace for mapping debugging purpose only and switch off trace for quality assurance (QA) and production environments.

Do not confuse the map trace file (.mtr) and the adapter trace file (specific per adapter). The adapter trace file is enabled with the **`-TRACE`** command on the command line of an adapter and generates a file, tracing the connectivity with the related application. While the adapter trace is not as critical as a map trace, it should be switched off in a production environment.

The trace file is created in the map's directory by default. A trace file has the .mtr extension. The location and name of the trace file can be changed in the Map Settings (Figure 7-10). You can set the MapTrace settings in the Map Editor or Integration Flow Designer. You can also use the Trace execution command (**-TI**) with the Command Server or the RUN ( ) function.

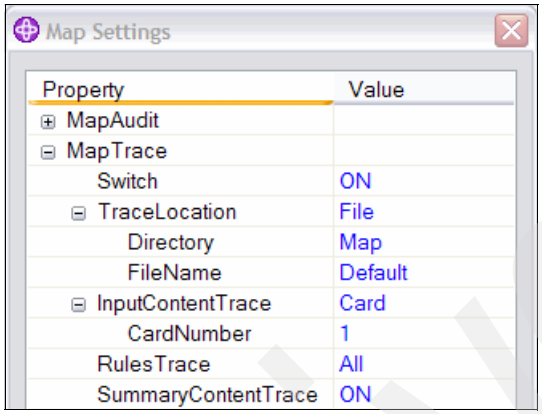

Figure 7-10 MapTrace settings with the InputContentTrace setup for input card #1

In a trace file, many elements are used to explain objects (Figure 7-11). Such elements include Level number, Offset number, len, comp, # number, DI number, data at offset number, and X' number. The DI and X' numbers are lab tools and are not relevant for WebSphere Transformation Extender developers. Ignore them when you review a trace file.

```
(Level 3: Offset 3, len 1, comp 1 of 4, #1, DI 00000001:)
Data at offset 3 ('*') was found to be of TYPE
   X'0011' (Element Delimiter Control ANSI EDI).

(Level 4: Offset 4, len 2, comp 1 of 15, #1, DI 00000002:)
Data at offset 4 ('00') was found to be of TYPE
   X'0014' (Auth'nInfoQual'r Element Control ANSI EDI).

(Level 4: Offset 7, len 10, comp 2 of 15, #1, DI 00000003:)
Data at offset 7 ('          ') was found to be of TYPE
   X'0015' (Auth'nInfo Element Control ANSI EDI).

(Level 4: Offset 18, len 2, comp 3 of 15, #1, DI 00000003:)
Data at offset 18 ('00') was found to be of TYPE
   X'0016' (SecurityInfoQual'r Element Control ANSI EDI).
```

Figure 7-11 Trace file excerpt

The Level <number> shows the level of the object in the card (Figure 7-12). The level of an object indicates the position of the data with respect to the entire card object. A card object has level 0. A component of the card object has level 1 and so on.

```
(Level 4: Offset 110, len 2, comp 1 of 8, #1, DI 00000016:)
Data at offset 110 ('PO') was found to be of TYPE
   X'0037' (Funct'lIDCd Element V4010 ANSI EDI).
```

Figure 7-12 Trace file highlighting the Level number

The Offset <position> is the offset of the data object in the data stream (Figure 7-13). This is the number of bytes from the beginning of the data that is passed to the input card.

```
(Level 4: Offset 110, len 2, comp 1 of 8, #1, DI 00000016:)
Data at offset 110 ('PO') was found to be of TYPE
   X'0037' (Funct'lIDCd Element V4010 ANSI EDI).
```

Figure 7-13 Trace file highlighting the Offset position

The len <number> is the length of the data object (Figure 7-14).

```
(Level 4: Offset 110, len 2, comp 1 of 8, #1, DI 00000016:)
Data at offset 110 ('PO') was found to be of TYPE
   X'0037' (Funct'lIDCd Element V4010 ANSI EDI).
```

Figure 7-14 Trace file highlighting the len number

The comp <number> of <number> element is the component number. For example, in Figure 7-15, it shows that the object is component 1 of 8.

```
(Level 4: Offset 110, len 2, comp 1 of 8, #1, DI 00000016:)
Data at offset 110 ('PO') was found to be of TYPE
   X'0037' (Funct'lIDCd Element V4010 ANSI EDI).
```

Figure 7-15 Trace file highlighting the comp number

The #<number> is the instance of the component (Figure 7-16).

```
(Level 4: Offset 110, len 2, comp 1 of 8, #1, DI 00000016:)
Data at offset 110 ('PO') was found to be of TYPE
   X'0037' (Funct'lIDCd Element V4010 ANSI EDI).
```

Figure 7-16 Trace file highlighting the # number

The next line shows “Data at offset <number> (‘<data>’), where the <data> is the first 15 characters of the actual data (starting at the listed offset position) (see Figure 7-17).

```
(Level 4: Offset 110, len 2, comp 1 of 8, #1, DI 00000016:)
Data at offset 110 ('PO') was found to be of TYPE
   X'0037' (Funct'lIDCd Element V4010 ANSI EDI).
```

Figure 7-17 Trace file highlighting the Data at offset

The data is followed by a message, such as the following examples (Figure 7-18):

- “is known to exist”
- “was found to be of TYPE”
- “is INVALID data of TYPE”
- “failed item presentation test for TYPE”
- “is required, but does not exist”
- “occurrence <number> is optional and has no content”
- “occurrence <number> is optional and does not exist”
- “does not match INITIATOR ‘<initiator>’”

```
(Level 4: Offset 110, len 2, comp 1 of 8, #1, DI 00000016:)
Data at offset 110 ('PO') was found to be of TYPE
   X'0037' (Funct'lIDCd Element V4010 ANSI EDI).
```

Figure 7-18 Trace file highlighting the trace message

The message is important because it indicates exactly what the input validation believes the object to be.

The <name> that follows the X’<number>, in parenthesis, is the type name to which the physical data presumingly belongs (Figure 7-19).

```
(Level 4: Offset 110, len 2, comp 1 of 8, #1, DI 00000016:)
Data at offset 110 ('PO') was found to be of TYPE
   X'0037' (Funct'lIDCd Element V4010 ANSI EDI).
```

Figure 7-19 Trace file highlighting the object name

These trace file elements can be used to determine where the data identified exists in the actual data file and where the item or group exists in the type tree, as seen in the Map Editor or Type Tree Editor. In Figure 7-20, the trace file points to an object at offset 110 for a length of 2, with a value of 'PO'.

```
(Level 4: Offset 110, len 2, comp 1 of 8, #1, DI 00000016:)
Data at offset 110 ('PO') was found to be of TYPE
   X'0037' (Funct'lIDCd Element V4010 ANSI EDI).
```

Figure 7-20 Input trace pointing to the input data

You see this item in the input data stream in Figure 7-21, starting at offset 110.

```
ISA*00*          *00*          *ZZ*AFFLINK        *ZZ*GPPPDP         *050224*1701*U*00401*000001697*0*P*>
GS*PO*AFFLINK*GPPPDP*20050224*1701*000000400*X*004010
ST*850*000000002
BEG*00*NE*9110477**20050224
```

Figure 7-21 Input data highlighting the data from the trace

Figure 7-22 shows the object listed in the trace file, which you can correlate to the object in the Map Editor (Figure 7-23) or the Type Tree Editor (Figure 7-24 on page 456).

```
(Level 4: Offset 110, len 2, comp 1 of 8, #1, DI 00000016:)
Data at offset 110 ('PO') was found to be of TYPE
   X'0037' (Funct'lIDCd Element V4010 ANSI EDI).
```

Figure 7-22 Input trace file pointing to the object in the type tree definition

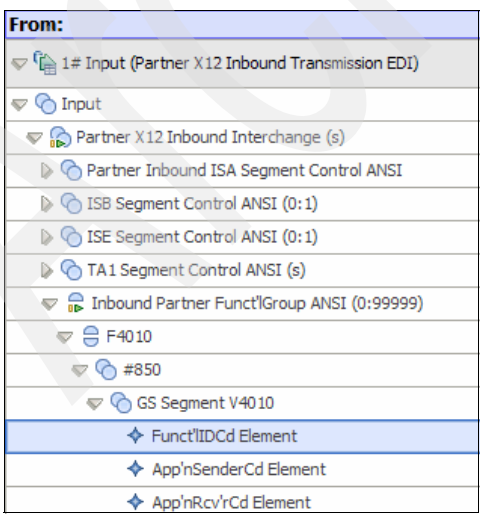

Figure 7-23 Map Editor view of an object from the trace file

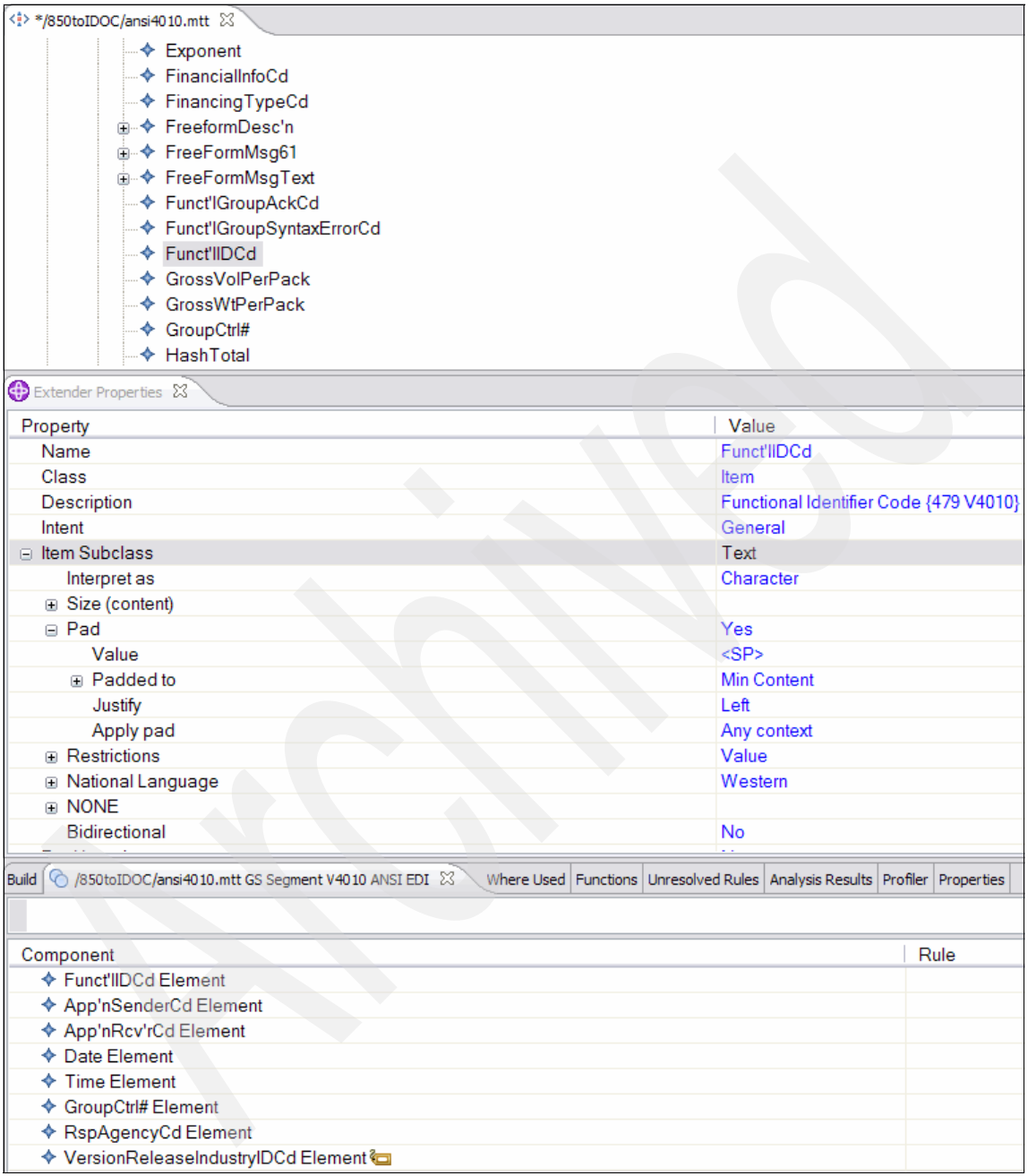

Figure 7-24 Type Tree Editor view of object from trace file

You can use trace messages to determine the existence of data. Certain properties can indicate whether an object exists prior to determining if it is invalid. If an object is invalid but known to exist, validation continues.

For example, the object `N1*BT*Distributor Co` is valid (Figure 7-25).

```
(Level 6: Offset 249, len 0, comp 1 of 3, #1, DI 0000002D:)
Type X'00E2' (N1 Segment V2003 ANSI EDI) is known to exist.

(Level 6: Offset 249, len 2, comp 1 of 3, #1, DI 0000002D:)
Data at offset 249 ('BT') was found to be of TYPE
   X'0055' (EntityIDCd Element V2003 ANSI EDI).

(Level 6: Offset 252, len 14, comp 2 of 3, #1, DI 0000002E:)
Data at offset 252 ('Distributor Co') was found to be of TYPE
   X'0061' (Name Element V2003 ANSI EDI).

(Level 5: Offset 248, len 18, comp 1 of 12, #1, DI 0000002F:)
Data at offset 248 ('*BT*Distributor ') was found to be of TYPE
   X'00E2' (N1 Segment V2003 ANSI EDI).
```

Figure 7-25 Valid object

However, the object `N1*++*Distributor Co` is invalid (Figure 7-26).

```
(Level 6: Offset 249, len 0, comp 1 of 3, #1, DI 0000002D:)
Type X'00E2' (N1 Segment V2003 ANSI EDI) is known to exist.

(Level 6: Offset 249, len 2, comp 1 of 3, #1, DI 0000002D:)
Data at offset 249 (‘++...') not in RESTRICTION LIST of TYPE
   X'0055' (EntityIDCd Element V2003 ANSI EDI).

(Level 6: Offset 249, len 2, comp 1 of 3, #1, DI 0000002D:)
Data at offset 249 (‘++') is INVALID data of TYPE
   X'0055' (EntityIDCd Element V2003 ANSI EDI).

(Level 6: Offset 252, len 14, comp 2 of 3, #1, DI 0000002E:)
Data at offset 252 ('Distributor Co') was found to be of TYPE
   X'0061' (Name Element V2003 ANSI EDI).
```

Figure 7-26 Invalid object

You can use trace messages to find unknown data. Unknown data also happens if additional data is found that does validate as part of the input (Figure 7-27). For example, garbage is at the end of a file or a file that contains several records, but the card only points to one record. (One record is read, and others are unknown additional data.)

Unknown data can also occur from not changing the default range on a multiple occurrence component, such as the record level of a file. The trace file validates the first records and then finds data that is undefined because the file is made of (1:1) a record.

```
(Level 0: Offset 0, len 507, comp 1 of 0, #1, DI 00000062:)
Data at offset 0 ('ISA*00*IBM CORP') was found to be of TYPE
   X'0005' (Partner X12 Inbound Transmission EDI).

  INPUT 1 was valid but contained 507 bytes of unknown data at the
end.

   End of Validation messages for INPUT CARD 1.

   End of Execution messages.
```

Figure 7-27 Unknown data

A trace file might also provide noncritical negative messages. These messages can occur when data is validated against a type to which it does not belong, which is in either of the following situations:

- There is an optional occurrence of an object that does not exist is in the data.
- An object belongs to a partitioned type.

If an occurrence of an object is optional, data that is read might have more than one possibility of being validated against a type. Consider the case where a file consists of an A record (1:3) and a B record. After one A record is found, the next data can be either another A or a B. The trace indicates that an occurrence is optional and does not exist (Figure 7-28 on page 459).

```
…
NTE**This is a header message
SHH*DD*001*930701
SHH*DD*001*930701
N1*BT*Distributor Co
…

…
(Level 4: Offset 284, len 0, comp 15 of 31, #4, DI 00000035:)
Data at offset 284 ('N1*BT*Distributo...') does not match INITIATOR
'SHH'
  of TYPE X'00A0' (SHH Segment V2003 ANSI EDI).

(Level 4: Offset 284, len 0, comp 15 of 31, #4, DI 00000035:)
COMPONENT number 15 of TYPE X'0042' (Transaction #850 Inbound
Partner Set V2003 ANSI EDI):
occurrence 3 is optional and does not exist.
…
```

Figure 7-28 Optional occurrence message

When using partitioned types, validation of an object that belongs to a partition starts from the top of the list of subtypes of the partition until data fully satisfies one of them. A message indicating an object failed to be of a certain subtype is not critical, unless data does not match any of the partitions or anything else after them. After a specific subtype of a partition is found, the partitioned group is also found. See Figure 7-29 and Figure 7-30 on page 460.

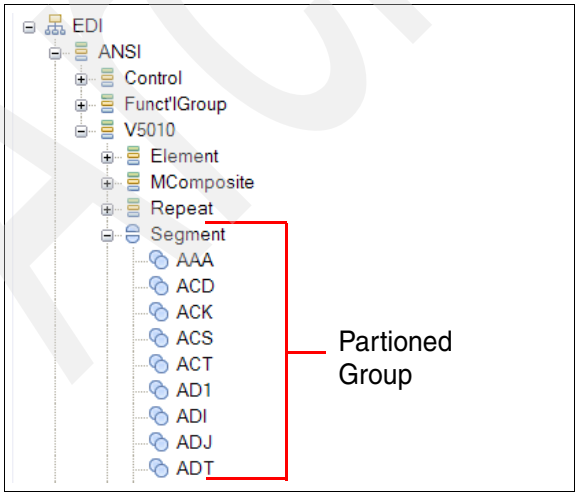

Figure 7-29 Partitioned group

Note: As explained in “Analyzing type trees” on page 68, it is particularly useful in this kind of structure to reorder subtypes (Addfirst or Addlast) and then sort partitions from the most frequent to the least frequent for performance reasons.

```
(Level 4: Offset 969, len 0, comp 1 of 12, #1, DI 000000C7:)
Data at offset 969 ('GE*2* <CR><LF>IEA*1*00...') failed COMPONENT
RULE test
   for component number 1, TYPE X'020C' (#819 Transaction IE Inbound
Partner Set V3060 ANSI EDI).

(Level 3: Offset 969, len 0, comp 2 of 3, #3, DI 000000C7:)
Data at offset 969 ('GE*2* <CR><LF>IEA*1*00...') failed PARTITIONing
for TYPE
   X'01C1' (Transaction IE Inbound Partner Set V3060 ANSI EDI).

(Level 3: Offset 969, len 0, comp 2 of 3, #3, DI 000000C7:)
COMPONENT number 2 of TYPE X'01BF' (IE F3060 Inbound Partner
Funct'lGroup ANSI EDI):
occurrence 3 is optional and does not exist.

(Level 4: Offset 972, len 0, comp 1 of 2, #1, DI 000000C7:)
Type X'0042' (GE Segment V3060 ANSI EDI) is known to exist.
```

Figure 7-30 Failed partitioning message

7.3 Additional troubleshooting techniques

A variety of additional troubleshooting techniques are available to try to solve problems in WebSphere Transformation Extender.

7.3.1 Summary Trace setting

The SummaryTrace setting is used to place a summary of the validation status of each input and the build status of each output after the map is built into the trace file (Figure 7-31).

```
INPUT 1 was valid.

End of Validation messages for INPUT CARD 1.

INPUT 2 was valid.

End of Validation messages for INPUT CARD 2.

INPUT 3 exists, but its type is in error.

End of Validation messages for INPUT CARD 3.


End of Execution messages.
```

Figure 7-31 SummaryContentTrace

Use the MapTrace settings (in Map Settings) in the Map Editor or Integration Flow Designer to turn on summary trace (Figure 7-32). Use can also use the Trace execution command (-TS) with the Command Server or the RUN () function.

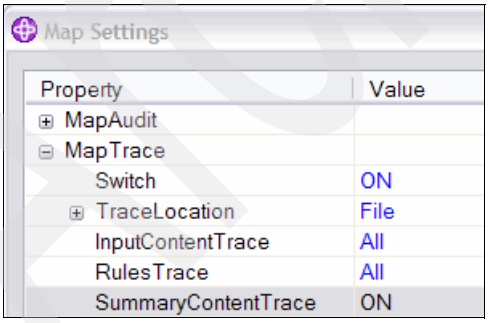

Figure 7-32 SummaryContentTrace setting

Note: SummaryContentTrace helps to locate the card that is invalid or contains errors.

7.3.2 Keywords

It is helpful to remember certain key words when you search through a trace file, log file, and so on. For example, the following keywords are useful to look for when debugging invalid data (Figure 7-33):

- Invalid
- Failed
- Not
- Wrong

Invalid is the most frequently found keyword, because any data discrepancy generates a validation issue and then generates the invalidation lines.

```
…
(Level 4: Offset 602, len 0, comp 24 of 30, #1, DI 00000072:)
Data at offset 602 ('CTT*3<CR><LF>SE*16*000') is INVALID data of
TYPE
   X'0208' (TDS Segment V3060 ANSI EDI).
…
(Level 3: Offset 109, len 51, comp 1 of 3, #1, DI 0000001B:)
Data at offset 109 ('*IN*3959910033*0...') failed COMPONENT RULE
test
   for component number 1, TYPE X'0044' (#850 F3060 Inbound Partner
Funct'lGroup…EDI).
…
(Level 5: Offset 254, len 0, comp 1 of 6, #1, DI 0000002B:)
COMPONENT number 1 of TYPE X'01D1' (LoopN11 IE810 IE Inbound Partner
Set V3060 ANSI EDI)
  is required, but does not exist.
…
```

Figure 7-33 Invalidation lines, failed component rule trace, and missing required object

7.3.3 Locating the last error and working backwards

It is often best to locate the last error and work backwards to find the source problem. Locate the negative message in the trace file and go backwards to identify from the trace the object that failed. Locate the object in both the map card and the type tree. Identify the last good group that was found prior to the negative message, and proceed from there.

7.3.4 Offset value

Observing where the Offset value stops increasing in the trace file can be used to help identify where the validation failure occurs. The Offset value can be used to determine where the invalid object begins. Use the Offset value and len (length) value to find and identify the data considered to be invalid.

7.3.5 Comparing valid and invalid data from a trace

For complex type trees, the input map trace when validating good data can be compared to the input map trace when validating bad data. The comparison can help identify what is normal. Then the validation failure can sometimes be more easily recognized in the abnormal section of the trace when validating the bad data (Figure 7-34).

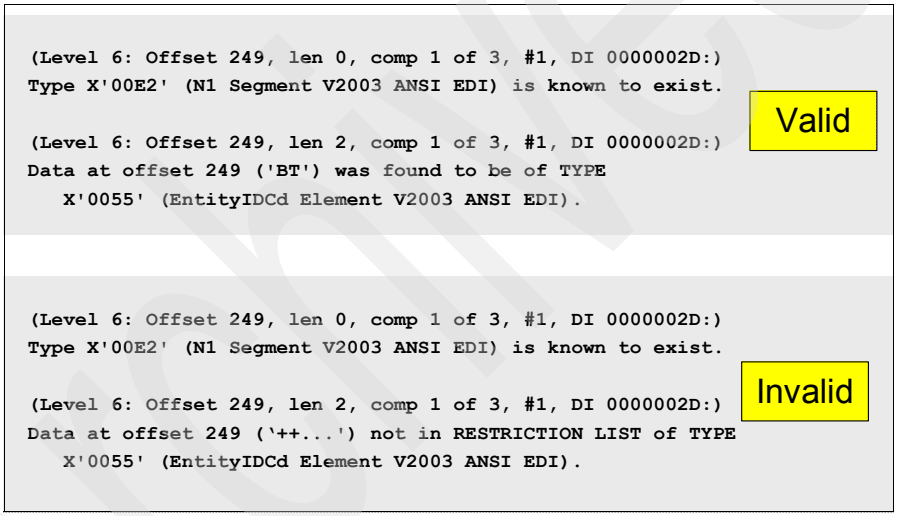

Figure 7-34 Example of comparing valid and invalid data

7.3.6 Map data audit log

You can specify that you want to audit a particular object by including that object in the Data Audit Settings view (Figure 7-35). To see the Data Audit Settings view, click **Window** → **Show View** → **Data Audit Settings**.

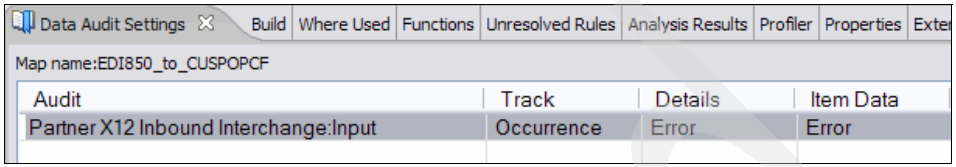

Figure 7-35 Data Audit Settings view

Data is only audited if the Map Settings window has the MapAudit → Switch set to ON and BurstAudit → Data is set to a value other than Never (Figure 7-36).

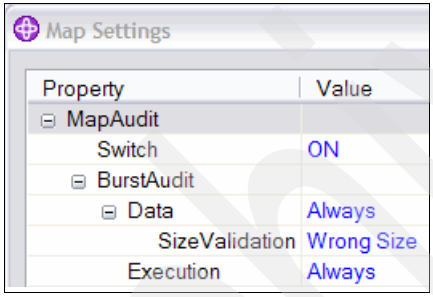

Figure 7-36 MapAudit settings

7.3.7 Error audit log status codes

The status attribute in the Audit Log is a letter followed by a code number. The letter of the status attribute indicates whether the data is valid (V), caused an error (E), or caused a warning (W). Table 7-1 explains the number in the status code if the letter is E.

Table 7-1 Audit Log status codes

Code	Description
E00	Object is an unidentified foreign object (UFO), which is data that is not associated with any particular type
E01	Object failed restriction
E02	Object failed presentation

Code	Description
E03	Object failed size check
E04	Invalid or missing initiator
E05	Invalid or missing terminator
E06	Object missing required component
E07	Object is invalid because it contains components in error
E08	Object failed partitioning
E09	Object failed component rule
E10	This required object is missing
E11	Invalid or missing delimiter
E12	More instances of an object exist than have been specified (for unordered groups only)
E13	Object failed to meet minimum size requirement
E14	Object failed to meet maximum size requirement

7.3.8 Map Debugger

The Map Debugger works as a classic debugging tool to help troubleshooting with output objects. The debugger can be used remotely or more frequently from the Design Studio. To begin, you activate the debug mode, then set up one or more breakpoints on every object that you want to monitor, and finally run the map. A debug window opens in which you see the construction of all objects with breakpoints. You can then use the Step In, Step Thru, Step Over, Continue, or Stop functions.

Note: Debugger mode does not affect a compiled map. It is handed by the Design Studio so that you do not need to recompile anything.

Activating the Map Debugger

To activate the Map Debugger, click the **Debug Map** icon (Figure 7-37) to switch on debugger mode.

Figure 7-37 Clicking the Debugger icon

Setting up breakpoints

By right-clicking any output object, you can choose from the following options (Figure 7-38):

- *Add Breakpoint*, which forces the transformation server to stop and show you the current object build. The object and its map rule is highlighted in yellow to show you that this object has a breakpoint (grey if object already has a breakpoint)
- *Delete Breakpoint*, which erases the selected object breakpoint and the transformation server will no longer stop to show construction of that object (grey if object has no breakpoint)
- *Delete All Breakpoints*, which erases all breakpoints in the current map
- *Show Breakpoints*, which shows all existing breakpoints in the current map

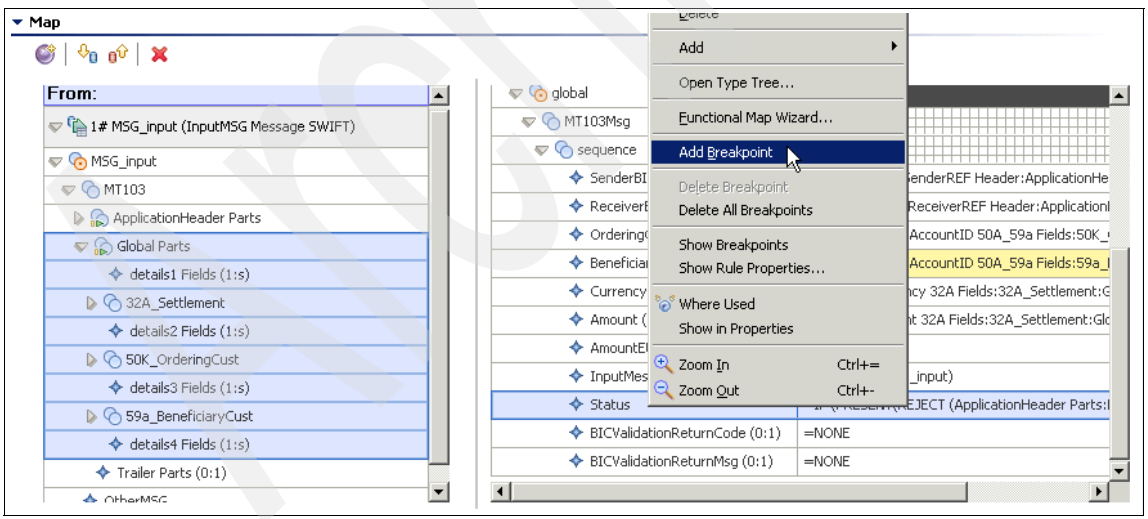

Figure 7-38 Setting up a breakpoint by using the pop-up menu

Then you can run the map. There is no need to compile because the debugger settings do not affect the compiled map. The debugger window opens and shows the object build for each breakpoint (Figure 7-39).

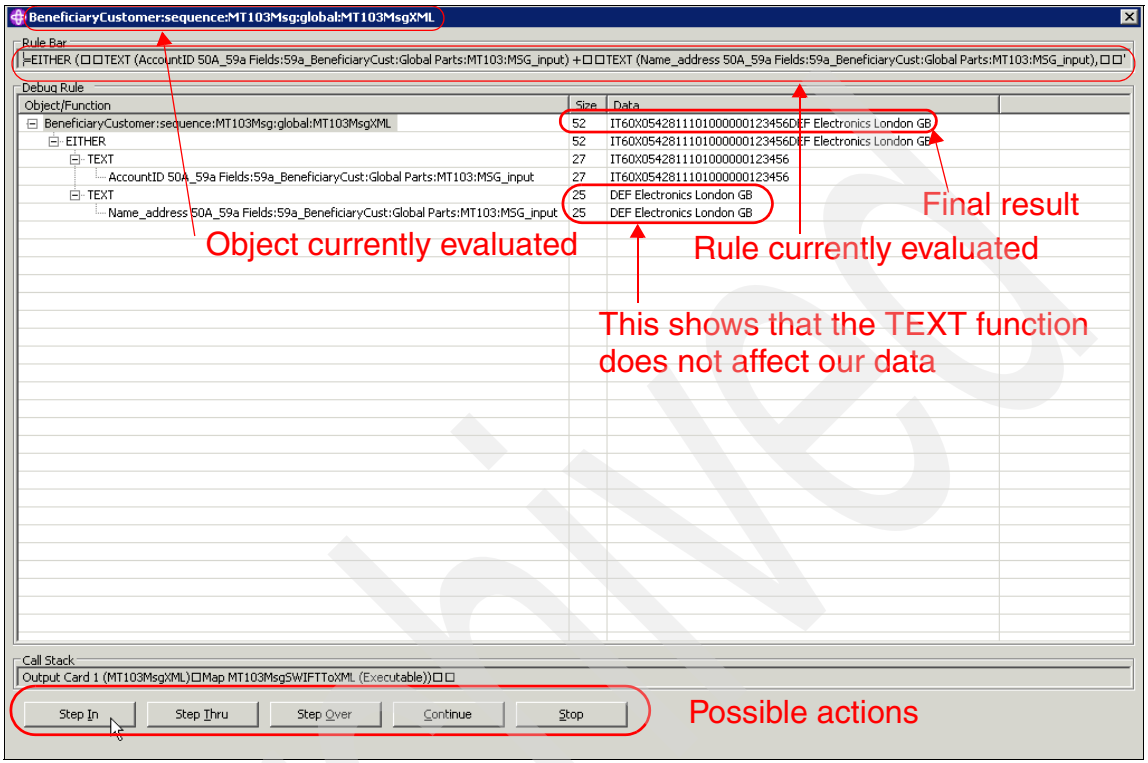

Figure 7-39 Debugger window showing an object build at breakpoint

7.4 Error handling

By default, WebSphere Transformation Extender stops the validation process for invalid data and does not produce any outputs. This is the cautious way of dealing with data transformation when a data discrepancy can be critical, such as in the healthcare industry. For example, you would not want process an incorrect prescription for a patient. Now for other business scenarios, stopping the transformation process and not producing data at all if any input data is invalid does not make any sense. In such a case, for example, a financial company that receives a batch of transaction orders from various brokers would go out of business if they decided to stop the transformation process of a file if one broker put invalid data on an order.

Therefore, the validation process can be adapted to the business scenario, allowing the transformation server to behave differently according to functional expectations.

7.4.1 Restart attribute

Assuming that you are transforming an input file with multiple records, you want to avoid stopping the transformation in case one record is invalid, ignore that record and all invalid records, and process all valid records.

To change the WebSphere Transformation Extender behavior, you simply add a Restart attribute to the record level of the type tree. You open the object that contains the record level and assign a Restart attribute to the Record object. You add a Restart by right-clicking the object and selecting **Restart**.

Figure 7-40 shows an example of assigning a Restart attribute to the ENR_ORDERSIN Record object.

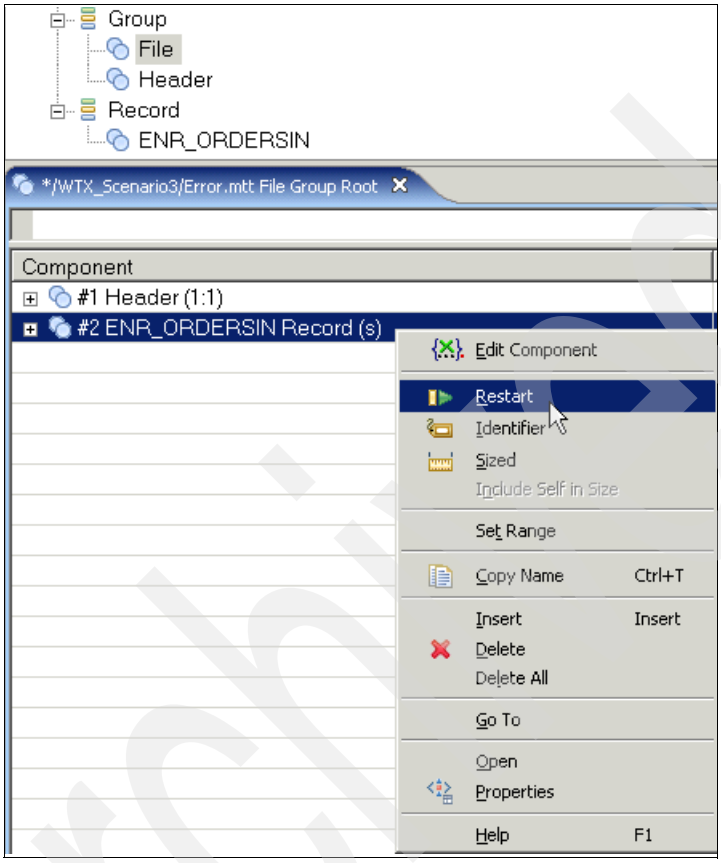

Figure 7-40 Setting up the Restart attribute on the record level

After doing assigning the Restart attribute, the record component of the top-level element file shows a Restart icon on both the Map Editor and Type Tree Editor view. You then analyze and save the type tree after the modification.

By using the Restart attribute, the transformation server restarts validation, if any invalid data is found, and proceeds with transformation for valid occurrences of the group that has the restart attribute.

The run result is completely different with Restart set, as shown in Figure 7-41.

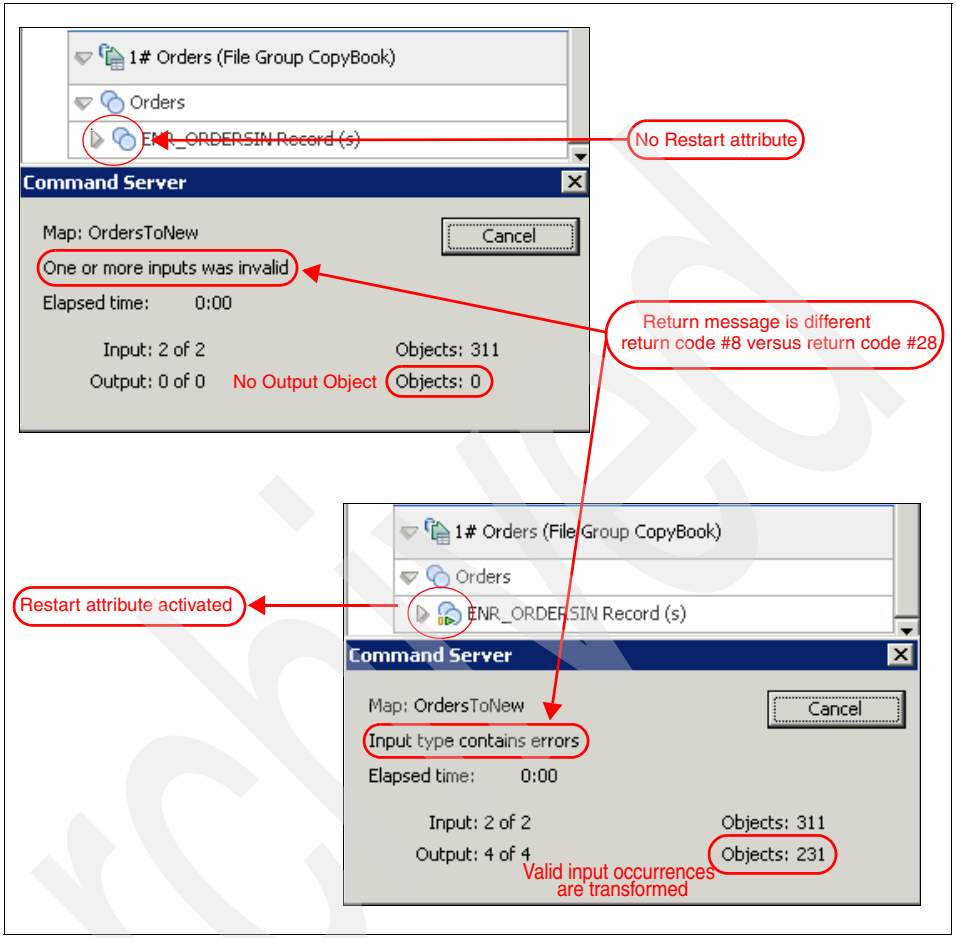

Figure 7-41 Different Run Result window depending on usage of the Restart attribute

The Restart attribute signals the WebSphere Transformation Extender engine to do the following tasks:

- Produce output for those objects that are not in error.
- Skip over objects it does not recognize or that contain errors in the data.

Without the Restart attribute, output is not produced if there are any validation errors.

7.4.2 REJECT function

The REJECT function is used in combination with the Restart attribute to access invalid occurrences of an input object. The REJECT function can only be used in a map rule.

The REJECT function has the following syntax:

```
REJECT (Object_with_restart_attribute)
```

It returns a binary large object (BLOB; literal with undefined size). Figure 7-42 illustrates the REJECT function using an input object with a Restart attribute.

Note: The argument of the REJECT function is an object with a Restart icon. It is faster to identify graphically.

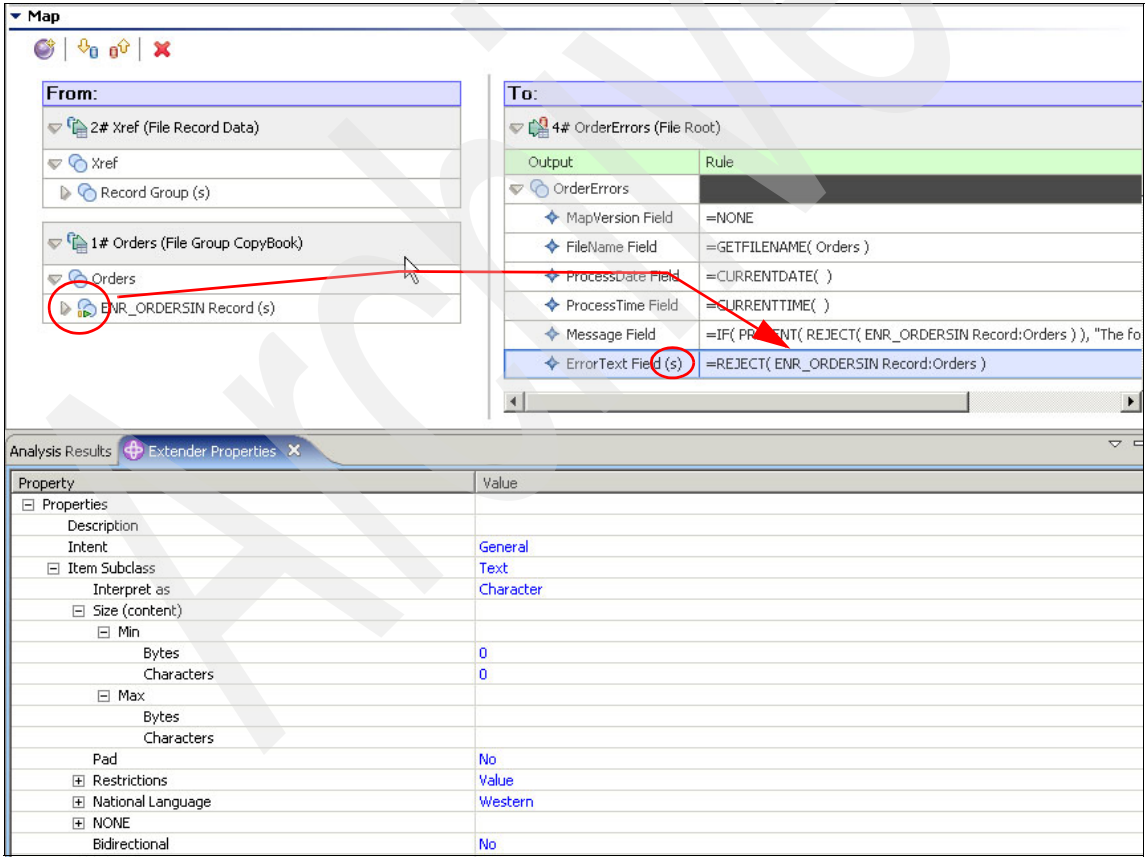

Figure 7-42 REJECT function used on a multiple Laurence BLOB, using input object with Restart attribute

You typically add a separate output card to the map to generate a file with the invalid input data. Any type tree can be used in which you have defined a text item with no specific size and with range set to (s), so that you can catch all occurrences of invalid data. For a concrete example, see Chapter 9, “Integration scenario: WebSphere Transformation Extender on System z” on page 623.

You can specify multiple levels of the Restart attribute and then manage each separately.

To use the Restart attribute and REJECT function:

1. Assign the Restart attribute to one or more components in an input type tree.
2. Design an error report.
3. Create an error report type tree.
4. Add an output card to an executable map for an error report.
5. Use the REJECT function.
6. Create functional maps as needed.
7. Create an output card for an error report.
8. Use the REJECT function component that has the Restart attribute.

Part 4

WebSphere Transformation Extender integration scenarios

In this part, we describe several integration scenarios. In the first scenario, we cover the use of WebSphere Transformation Extender for Integration Servers with WebSphere ESB. We then take the first scenario and expand upon it in a second scenario by using WebSphere Transformation Extender for Integration Servers with WebSphere Message Broker. Finally, we provide a scenario that uses WebSphere Transformation Extender on System z.

This part includes the following chapters:

- Chapter 8, “Integration scenarios: WebSphere Transformation Extender for Integration Servers” on page 475
- Chapter 9, “Integration scenario: WebSphere Transformation Extender on System z” on page 623

8

Integration scenarios: WebSphere Transformation Extender for Integration Servers

In this chapter, we describe and build two specific scenarios that use WebSphere Transformation Extender for Integration Servers and the WebSphere Transformation Extender Pack for SWIFTNet FIN. Both scenarios are built on a single business scenario that involves a credit transfer between two international companies. In each scenario, we use WebSphere Transformation Extender to interpret and validate the SWIFT message. The distinction is that, in the first scenario, we build the scenario on WebSphere ESB and use WebSphere Transformation Extender with the Pack for SWIFTNet FIN to interpret the message and transform it in an XML format that is more easily understandable by the ESB.

The second scenario is an extension of the one that we implement in scenario 1. Again we use WebSphere Transformation Extender to interpret and validate the SWIFT message. However, in this second scenario, WebSphere Message Broker is used to perform the Bank Identifier Code (BIC) validation, similar to WebSphere ESB in scenario 1. Functionality is also incorporated to perform additional validation checks against predefined criteria.

8.1 Business scenario introduction

Our business scenario to show the integration in WebSphere ESB and WebSphere Message Broker involves a credit transfer between two international companies. Company A wants to pay an invoice to Company B as shown in Figure 8-1.

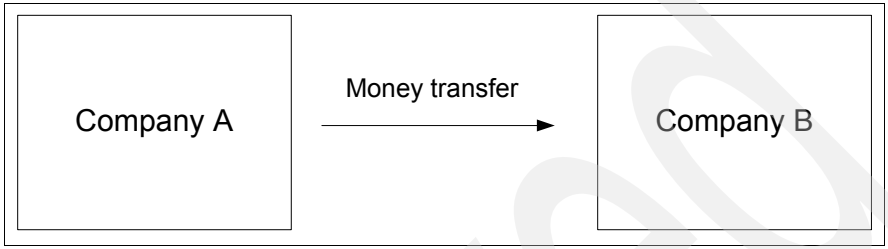

Figure 8-1 The high-level business scenario

Typically the companies never give each other cash money. Instead, they use their respective banks as intermediaries. For example, Company A instructs its bank to make a payment. This bank then performs a money transfer to Company B's bank. These money transfers between the banks in different countries typically pass over a special network for financial transactions, called Society for Worldwide Interbank Financial Telecommunication (SWIFT). Figure 8-2 shows how the banks and the SWIFT network participate in the payment process.

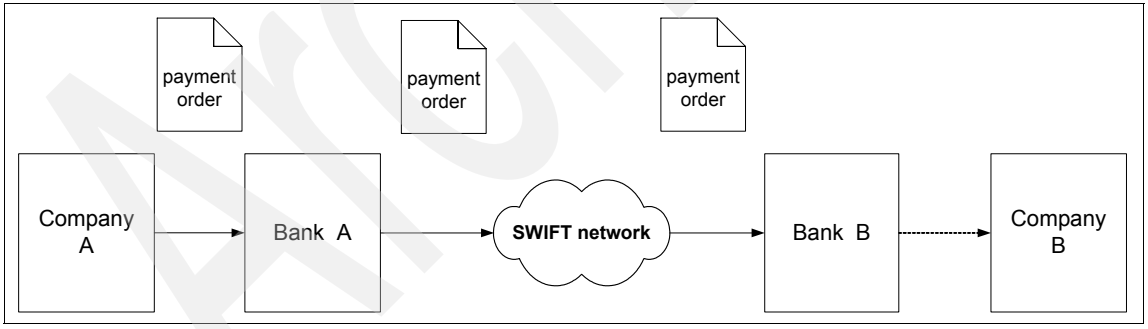

Figure 8-2 The business scenario with SWIFT network

Besides being a networking system, SWIFT also defines specific message formats for financial transactions. A payment instruction as used in this scenario is also known as the SWIFT MT103 message. For more information about this message, see 8.1.3, "The MT103 message" on page 480.

In reality the data flow for a financial transaction also includes several acknowledgements. However for simplicity, we do not go into more detail about acknowledgements for this scenario.

8.1.1 The process for scenario 1: WebSphere ESB

Figure 8-3 illustrates the passage of the payment message through the enterprise service bus (ESB) of bank A that is implemented in scenario 1.

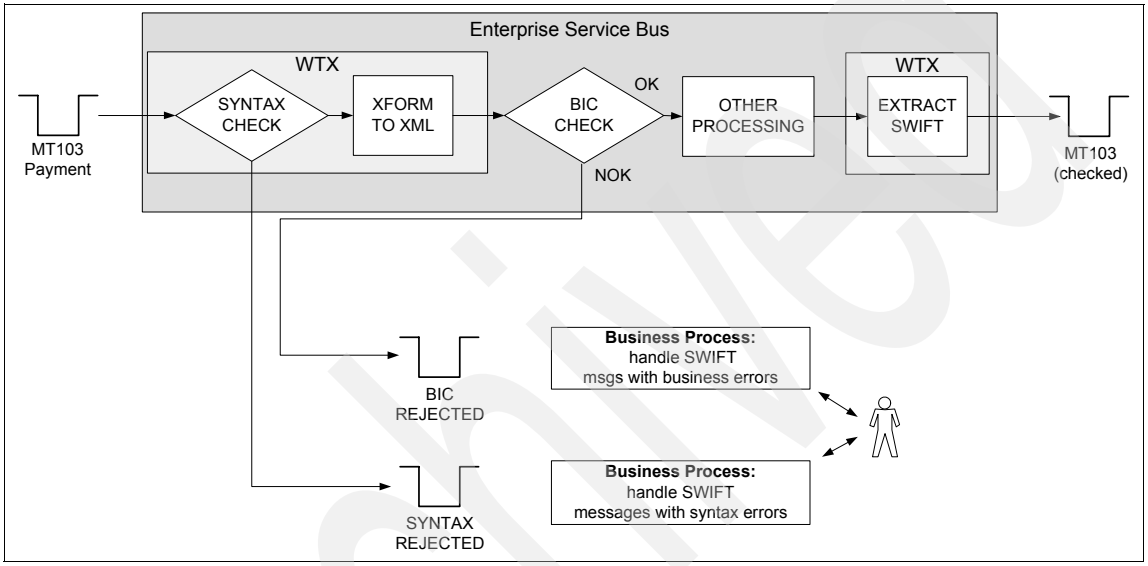

Figure 8-3 The scenario in WebSphere ESB

The SWIFT MT103 message comes from the front-end system. Our responsibility is to ensure that the message has the correct syntax and content before we send it to the back-end systems and the SWIFT network.

The ESB uses WebSphere Transformation Extender with the Pack for SWIFTNet FIN to interpret the message and transform the message in an XML format that is more easily understandable by the ESB. This XML format also contains the entire original SWIFT message in one of its fields. WebSphere Transformation Extender checks whether the payment request has a correct syntax. In this case, it completes the XML fields with values that it retrieves from the SWIFT message. In the case of an invalid syntax, it sets one of the XML fields to indicate that the message is incorrect and cannot be processed.

Type tree: In the lab, we use a simplified version of the type tree in the pack for SWIFT.Net FIN. However, Appendix C, “Using type trees from the SWIFTNet FIN industry pack” on page 735, describes how to do this with the complete type tree.

In the case of a correct syntax, the ESB performs additional validation steps on the content of the message. For example, a validation that we implemented is whether the BIC of the receiver is valid and connected to the SWIFT network. This validation is implemented in a Web service.

Note: In a real-world situation, you might perform more validation checks. However, for simplicity of the scenario, we limit the validation to just the BIC check.

If both the syntax check and BIC check are valid, the payment is forwarded to the back-end systems for interbank payment processing. A second map is used to extract the original MT103 message from the XML wrapper to send it out.

If either the syntax or the BIC is invalid, the payment order enters a business process to take the appropriate actions. This can be a process implemented in WebSphere Process Server, with human interaction. However, we do not implement any of the WebSphere Process Server functionality.

8.1.2 The process for scenario 2: WebSphere Message Broker

Figure 8-4 on page 479 illustrates the flow that we implement in scenario 2. The functionality is an extension of the one that we implement in scenario 1. Furthermore, we use WebSphere Message Broker in this scenario.

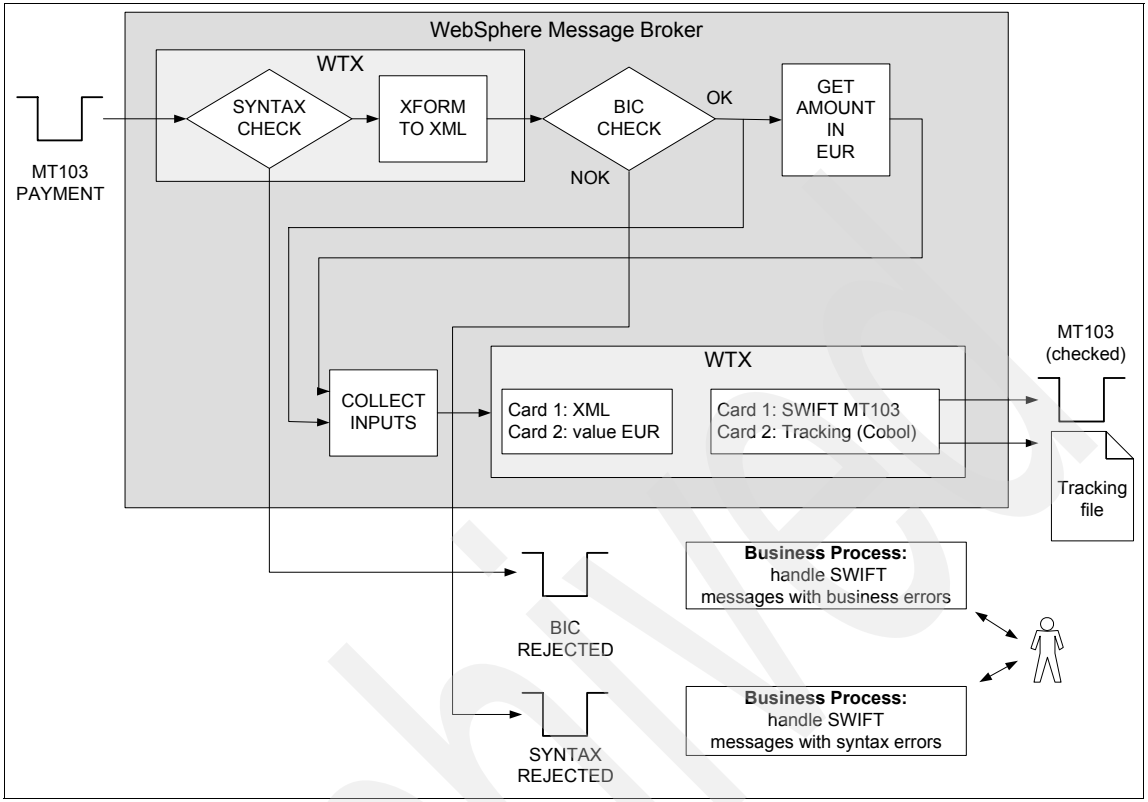

Figure 8-4 The scenario in WebSphere Message Broker

Again, a SWIFT message comes in. In this scenario, WebSphere MQ is used as a transport layer. Again WebSphere Transformation Extender is used to interpret and validate the SWIFT message. It sends an XML message back to WebSphere Message Broker. WebSphere Message Broker performs the BIC validation, similar to WebSphere ESB in scenario 1.

However, WebSphere Message Broker performs an additional check. That is, if the value of the transaction, calculated in euro (€), is larger than €10,000, the bank wants to maintain a trace of the transaction. Therefore, a Web service is used that converts the amount to euro. If the amount in euro that is returned by the Web service is larger than this threshold, a tracking file is created. In our implementation, we route the result of the Web service and the internal XML format into a second WebSphere Transformation Extender map. This map creates the outgoing SWIFT message and the tracking message.

Note: The tracking message can also be produced by the WebSphere Message Broker. However, we chose to implement this functionality in WebSphere Transformation Extender to show how to integrate a map with multiple inputs and multiple outputs into WebSphere Message Broker.

8.1.3 The MT103 message

The SWIFT MT103 message is a single customer credit transfer message that is used to transfer money between customers of different banks or financial institutions.

Figure 8-5 illustrates a typical situation where customer A wants to send money to customer B. The MT103 message is used twice in this example: first when bank A sends the information to SWIFT and second when SWIFT sends the information to bank B.

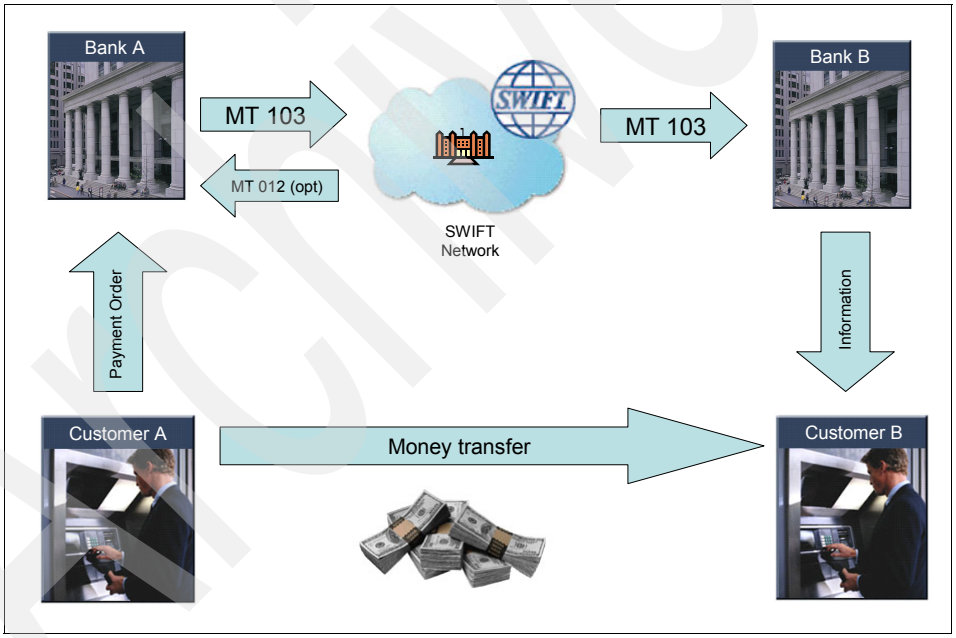

Figure 8-5 An example of using a MT103 SWIFT message

A typical situation flows as follows:

1. Customer A sends a bank a payment order.
2. Bank A prepares an MT103 message and sends it to the SWIFT network.
3. The SWIFT network sends an MT103 message to bank B.
4. Optional: The SWIFT network sends an acknowledgement to bank A by using a system message MT012.
5. Bank B informs the customer of the transfer.

SWIFTNet FIN messages are structured in blocks. Each block of a message contains a special type of data. Each block begins with a left brace ({) and ends with a right brace (}). The first two characters in a block are its number and the separator (:).

Therefore, a SWIFT message has the following structure:

- {1: Basic Header Block}
- {2: Application Header Block}
- {3: User Header Block}
- {4: Text Block}
- {5: Trailer}

Example 8-1 shows a sample MT 103 message.

Example 8-1 Sample MT103 message

```
{1:F01AAAAUS33XXX1234000123}{2:I103AAAAGB2LXXXXN}{3:{103:TGT}{113:NNNN}
}{4:
:20:BBBB-060928-CCT4
:13C:/CLSTIME/0915+0100
:23B:CRED
:32A:060929EUR10,2
:33B:EUR21,45
:50K:/ACC123
Name1
Name2
:52A:TESTBIC1
:56D:/INTRMDYAGT1002
56D LINE 1
56D LINE 2
56D LINE 3
56D LINE 4
:57A:/CDTRAGTACCT004
TESTBIC5
:59:/IT60X0542811101000000123456
```

```
DEF Electronics London GB
:70:ABC/4562/2006-09-08
:71A:BEN
:71F:EUR3,45
:71F:EUR4,56
-}
```

In all the SWIFT messages, we use BICs to identify the financial institutions that are participating in the operation.

Example 8-1 on page 481 has the following BICs:

- AAAAUS33XXX for the sending institution
- AAAAGB2LXXX for the receiving institution

8.1.4 Web services descriptions

In our integration scenarios, we use several Web services to perform external operations.

The BIC Validation Service

The BIC Validation Service is responsible for validation of the BIC. We send the service a receiver BIC to check if it is a valid code and can be used in the SWIFT network. Table 8-1 shows the possible replies from the BIC Validation Service.

Table 8-1 Possible replies from the BIC Validation Service

Return code	Description
0	This BIC is connected to the SWIFT network.
-1	This BIC is invalid.
-2	This BIC has been deleted from the last update and cannot be used any longer.
-3	This BIC is not connected to the SWIFT network and cannot be used in the header of a SWIFT message.

In our scenarios, we send an incoming message for further processing only if the BIC Validation Service returns a value of 0.

The Currency Conversion Service

The Currency Conversion Service is a sample Web service that converts amounts of money from different currencies to the equivalent amount in another currency. For input, the service takes an amount and a currency. The service

performs the translation and returns the amount in the new currency. We use this service to transform the original amount in euros for the scenario with WebSphere Message Broker.

We developed a basic version of this service, for simplicity reasons. As input currency, the service accepts U.S. dollars ($), British pounds (£), and euros. As target currency, only euros are supported.

8.2 Available resources and the components that are built in each scenario

First we provide background on the resources that are available to build the scenarios and what you can build yourself.

8.2.1 Available resources

This section describes the resources that are available to help you build the scenario.

WebSphere ESB scenario

Several sources are available for the WebSphere ESB scenario.

Software

In the WebSphere ESB scenario, we use the following software to develop maps and the application:

- WebSphere Integration Developer v6.1
- WebSphere Enterprise Service Bus v6.1
- WebSphere Transformation Extender Design Studio v8.2.0.3
- WebSphere Transformation Extender for Integration Servers v8.2.0.3

> **Note:** In a real-world project, also use the WebSphere Transformation Extender Pack for SWIFTNet Finance. However, in our scenario, we use a simplified version of the SWIFT MT103 message type tree.

Verify that all of the necessary prerequisites are installed properly before you start to follow the scenario steps.

Installed prerequisites: We assume that all the prerequisites are installed on the same machine. In a real-world project situation, you must have installed WebSphere Transformation Extender Design Studio and WebSphere Integration Developer on the development platform. You must also have installed WebSphere ESB and WebSphere Transformation Extender for Integration Servers in the runtime environment.

Downloadable prerequisites

In one part of the scenario, we intend to call an external service to validate the receiver BIC. A sample implementation of this service has been prepared and is available for download. See Appendix E, “Additional material” on page 749, for details. Also download the sample MT103 messages, which are required to test the scenario.

WebSphere Message Broker scenario

We build the scenario by using WebSphere Message Broker v6.1.0.2 and WebSphere Transformation Extender for Integration Servers v8.2.0.3.

The WebSphere Message Broker scenario is partially prebuilt. However, the WebSphere Transformation Extender nodes are not available in the prebuilt solution. Our goal is to show how these maps can be added to complete the flow. We start from a WebSphere Message Broker default configuration.

Figure 8-6 shows the entire list of projects that you see after you import the Project Interchange file into the Message Broker Toolkit.

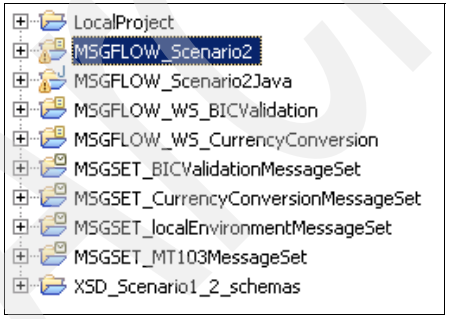

Figure 8-6 The WebSphere Message Broker projects

In the following sections, we briefly explain the content of each of these projects.

LocalProject

LocalProject is not part of the content of the Project Interchange file, but it is created when you run the WebSphere Message Broker Default Configuration Wizard. This project contains the broker domain information.

MSGFLOW_Scenario2

The MSGFLOW_Scenario2 project is the main message flow project (Figure 8-7 on page 485). It contains the flow (Scenario2) that we complete by adding the WebSphere Transformation Extender maps. The place where these maps belong has been marked with Passthru nodes.

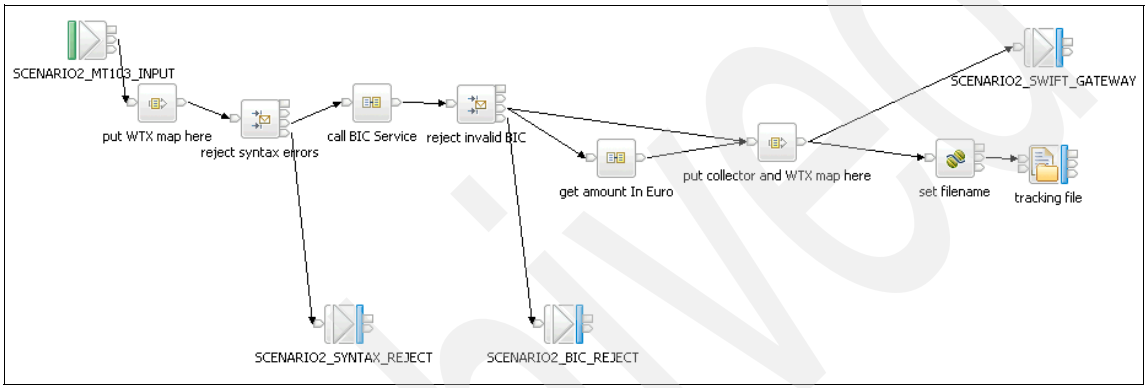

Figure 8-7 The message flow

The message flow has the following functionality:

- The *SCENARIO2_MT103_INPUT* MQInput node reads messages from the equally named WebSphere MQ queue. It does not parse the content of the message but reads it in as a binary large object (BLOB).
- The *put WTX map here* Passthru node is where we put our first Transformation Extender map. You delete this node and replace it with a WTX Map node. The WebSphere Transformation Extender map parses and validates the SWIFT message and creates an XML message for treatment further downstream in the flow. The XML message is defined by the MT103Msg message in the message set MT103MessageSet.
- The *reject syntax errors* Routing node reads the XMLNSC tree structure that is created by the WTX node and checks whether it has a status of OK. If so, it routes the message to its OK output terminal. If not, the message is routed to the SYNTAX_ERROR output terminal.
- The *SCENARIO2_SYNTAX_REJECT* MQOutput node writes all messages for which a syntax error has been detected to the equally named WebSphere MQ queue.

- The *call BIC service* node represents a subflow (`Subflow_callBICService`, shown in Figure 8-8), which maps the internal XML structure to the BIC Web service request, executes the Web service call, and adds the result of the Web service call back to the XMLNSC tree structure.

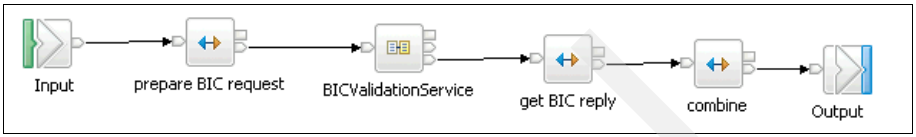

Figure 8-8 The subflow behind the call BIC service node

- The *reject invalid BIC* router node checks the result of the BIC Validation Web service. If the BIC is okay, the node routes the message to its BIC_OK terminal for further treatment by the rest of the flow. If a BIC error is returned by the Web service, the message is routed to the BIC_NOK output terminal.

Note: Messages coming out of the BIC_OK terminal are routed in parallel to the *put collector and WTX map here* node and to the *get amount in Euro* node. The output of the *get amount in Euro* node is then also routed to the *put collector and WTX map here* node. This means that the second WebSphere Transformation Extender map receives two input cards:

- One card to receive the XMLNSC message that is propagated through the whole flow
- One card to receive the result of the currency conversion Web service

- The *SCENARIO2_BIC_REJECT* MQOutput node writes all messages with a failed BIC check to the equally named WebSphere MQ queue.
- The *get amount in Euro* node again represents a subflow. It points to Subflow_getAmountInEuro, which is shown in Figure 8-9. This subflow maps the XML structure to the Web service request and passes the entire content of the Web service reply (without the SOAP header) back to the main flow and into the Transformation Extender node.

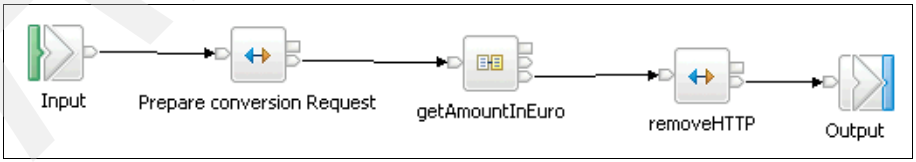

Figure 8-9 The subflow behind the 'get amount in Euro' node

- The *put collector and WTX map here* node is a temporary passthru node that you delete and replace by a WTX Map node and a Collector node that allow multiple inputs for the WTX Map node.
- The *SCENARIO2_SWIFT_GATEWAY* MQOutput node receives the SWIFT messages from the WTX Map node and places them on the WebSphere MQ queue with the same name. These messages are then sent to the back-end systems for further processing.
- The *set filename* Java compute node performs two tasks:
  - It generates a file name for the tracking files that we create when the amount of the transaction is higher than €10,000.
  - It checks whether the message coming from the previous node has data in its body. If the body is empty, it discards the message and does not propagate it. Figure 8-10 shows the lines of Java code that remove the empty messages.

```
// if the message is empty we do not propagate the message
MbElement blob = inMessage.getRootElement().getLastChild().getLastChild();
byte[] ba = (byte[])blob.getValue();
if (ba.length > 0) {
    out.propagate(outAssembly);
}
```

Figure 8-10 Java snippet to discard empty messages

Create on Content feature: We check whether the message has content because we use the *Create on Content feature* in WebSphere Transformation Extender. When setting this property on a WebSphere Transformation Extender output card, the output data for this card is only sent if some content for the data exists. The map does not send empty output data if it has this setting.

In our case, we use Create on Content feature to instruct the map to only create data for the tracking file if the transaction amount is higher than €10,000. However, in the current release of WebSphere Transformation Extender for Integration Servers (8.2.0.3), the WTX Map node overrides this property. Therefore, if the transaction amount is less than €10,000, and no tracking file has to be created, the WebSphere Message Broker flow still receives a message with an empty body instead of no message at all. This is why we use this snippet of Java code to discard empty messages.

- The *tracking file* FileOutput node creates the tracking file with the file name that is provided by the JavaCompute node. The created files are placed in the `C:\$Redbook\WMB_Scenario` directory.

MSGFLOW_Scenario2Java

The MSGFLOW_Scenario2Java project contains some Java code that is used by the *set filename* node in the main flow.

MSGFLOW_WS_BICValidation

The MSGFLOW_WS_BICValidation project contains the implementation of the BIC Validation Web service.

MSGFLOW_WS_CurrencyConversion

The MSGFLOW_WS_CurrencyConversion project contains the implementation of the Currency Conversion Web service.

MSGSET_BICValidationMessageSet

The MSGSET_BICValidationMessageSet project contains the message definitions that are used by the BIC Validation Web service.

MSGSET_CurrencyConversionMessageSet

The MSGSET_CurrencyConversionMessageSet project contains the message definitions that are used by the Currency Conversion Web service.

MSGSET_localEnvironmentMessageSet

The MSGSET_localEnvironmentMessageSet project contains a message definition for you to use visual mapping nodes in order to copy data to and from the LocalEnvironment memory space.

MSGSET_MT103MessageSet

The MSGSET_MT103MessageSet project contains the message definition of the XML format that is handed off by WebSphere Transformation Extender to WebSphere Message Broker.

XSD_Scenario1_2_Schemas

The XSD_Scenario1_2_Schemas project contains the XML schema files that we used to create the WebSphere ESB and WebSphere Message Broker scenarios.

8.2.2 The components that are built in each scenario

For the scenarios, we build some custom components and show how to configure other WebSphere Transformation Extender generated components to complete the scenario, which includes building an ESB module. Then we walk through the tasks of creating, completing, and configuring maps; working with a simplified type tree; configuring adapters; and adding map nodes.

WebSphere ESB scenario

In the WebSphere ESB scenario, we build a small WebSphere ESB module. The module uses the WebSphere Adapter for Flat Files and WebSphere Transformation Extender data binding to read a SWIFT MT103 message and convert it to the business object.

We show how to perform the following tasks:

- Create WebSphere Transformation Extender maps for WebSphere ESB and WebSphere Process Server by using the Map Generator for WebSphere ESB or WebSphere Process Server.
- Prepare a simplified type tree for the SWIFT MT103 message. Note that, in a real-world project, you use the SWIFTNet Fin pack for this.
- Complete the maps generated by the Map Generator.
- Configure the WebSphere Flat File Adapter to use the WTX data binding.

We also create a simple mediation flow. This flow calls an external Web service and, based on its reply, routes the message to different nodes.

WebSphere Message Broker scenario

We start with the message flow as shown in Figure 8-7 on page 485. In this message flow, we replace the Passthru nodes with real WTX Map nodes. We show how you can perform the following tasks:

- Add a simple WTX Map node that reads one input from the message flow and sends one output back to the message flow.
- Add a more complex WTX Map node that feeds two input cards from the message flow (by using the Collector node) and passes the data from two output cards back into the message flow. Figure 8-11 shows the final flow that we create.

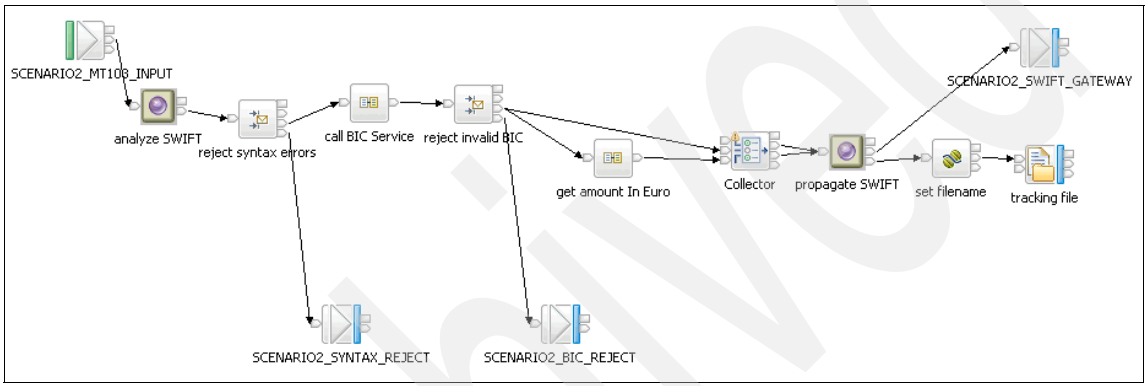

Figure 8-11 The complete WebSphere Message Broker flow

8.3 Building the WebSphere ESB scenario

In building the WebSphere ESB scenario, we explain the step-by-step instructions on how to create WebSphere Transformation Extender maps and an ESB application that uses the WebSphere Transformation Extender maps.

8.3.1 Roadmap for building the WebSphere ESB scenario

Figure 8-12 illustrates the steps to complete the scenario.

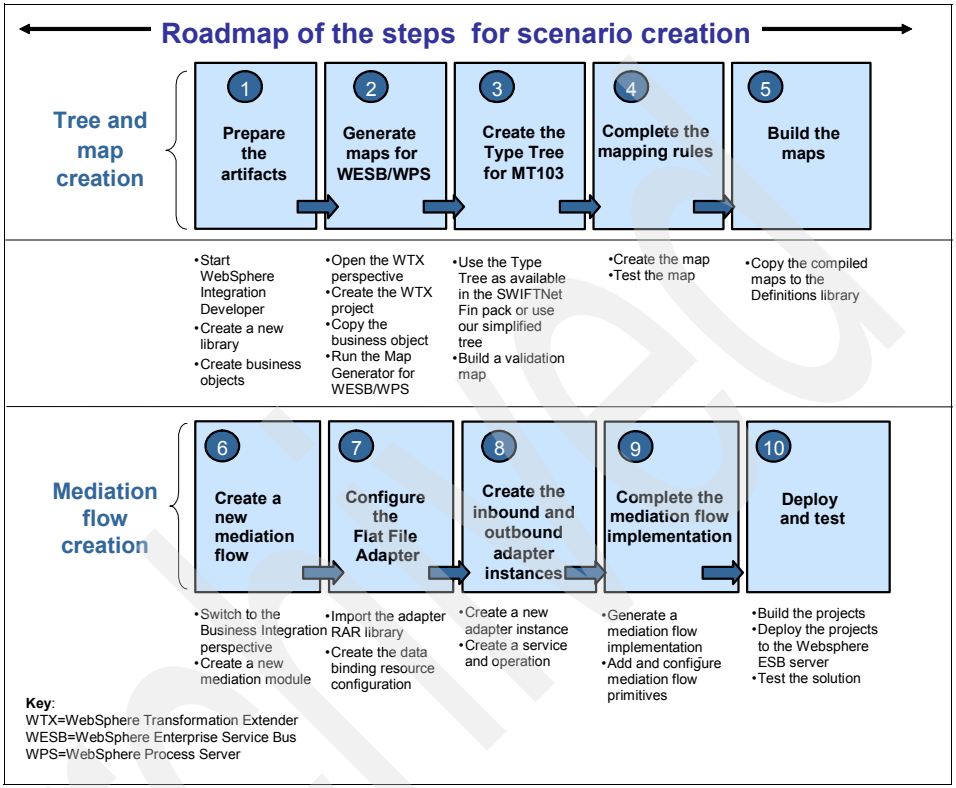

Figure 8-12 Roadmap of the steps for creating the scenario

8.3.2 Preparing the artifacts

Before we start to develop the maps to transform the MT103 message into the business object, we must create the necessary artifacts. To create the library that we use later in both the WebSphere ESB and WebSphere Message Broker scenarios:

1. Start the IBM WebSphere Integration Developer.
2. Switch to the Business Integration Perspective. Select **Window** → **Open Perspective** → **Business Integration**.
3. Create a new library. Select **File** → **New** → **Library**, and name the library `Definitions`.

4. Create a new business object:
   a. In the Definitions library, under Definitions, right-click **Data Types** and select **New** → **Business object**.
   b. In the New Business Object window (Figure 8-13), for Name, type `MT103Msg`, and for Namespace, enter `http://www.ibm.com/Redbooks/MT103Msg`. Then click **Finish**.

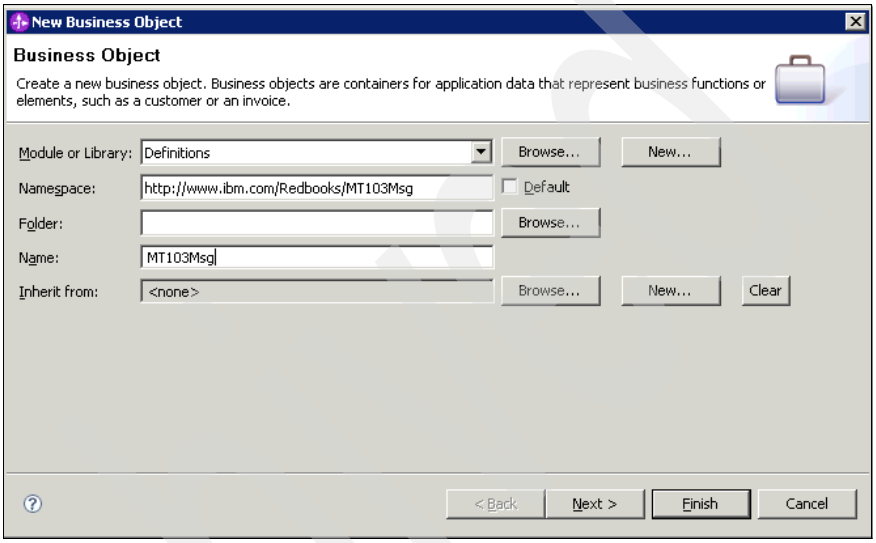

Figure 8-13 New Business Object window

5. Complete all the fields for the business object (under MT103Msg in the upper pane in Figure 8-14).

   In our scenarios, we use only a few fields from the original MT103 message. Additionally, in the InputMessage field, we store the whole incoming MT103 message in its native format. After completing the whole process, we use the InputMessage field to send the MT103 message in its original form for further processing.

   The map that converts the MT103 message into a business object sets the Status field with the `SYNTAX ERROR` value whenever the validation of the incoming message fails. Otherwise this field is set to `OK`.

   Then save and close the business object.

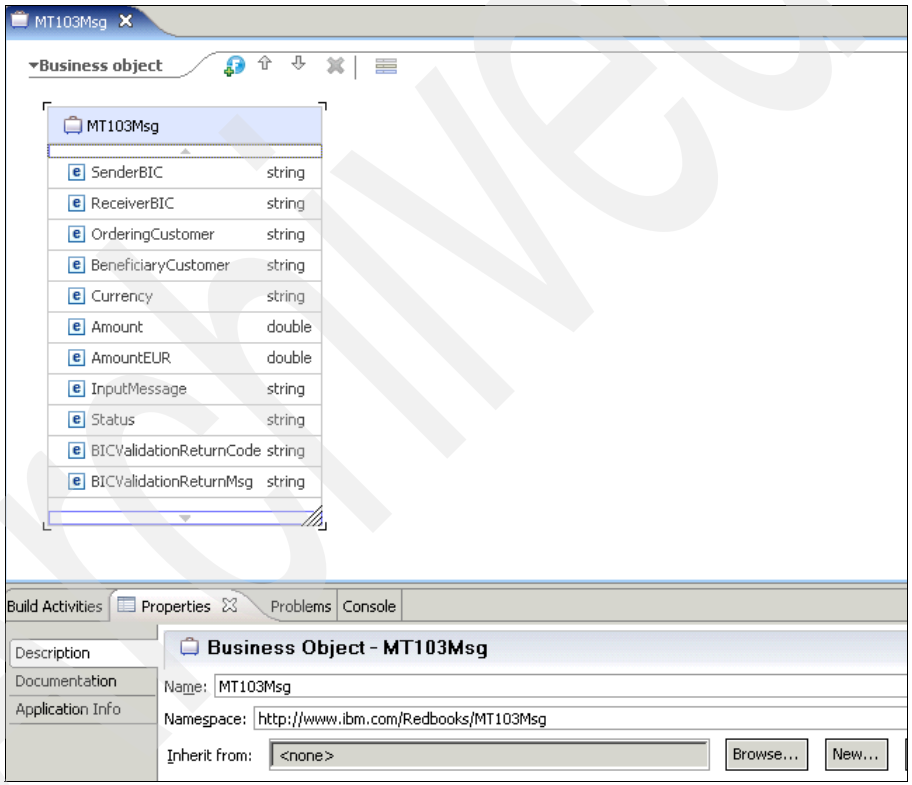

Figure 8-14 MT103Msg business object

6. Repeat steps 4 on page 492 and 5 on page 493 for creating the business objects for the BIC Validation service:

   a. Create the BICValidationRequest business object in the `http://www.ibm.com/Redbooks/BICValidation` namespace (Figure 8-15).

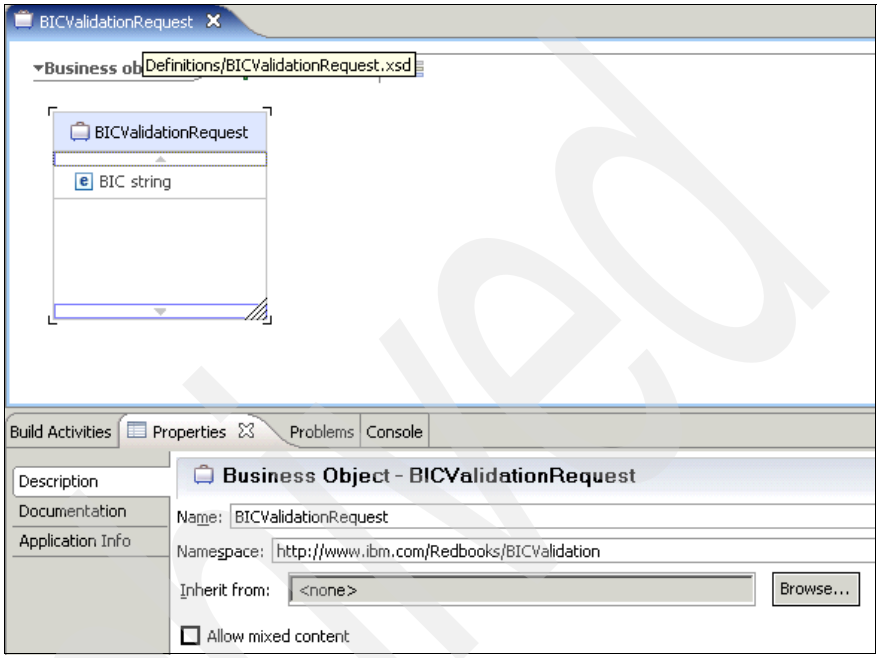

Figure 8-15 BICValidationRequest business object

b. Create the BICValidationReply business object in the `http://www.ibm.com/Redbooks/BICValidation` namespace (Figure 8-16).

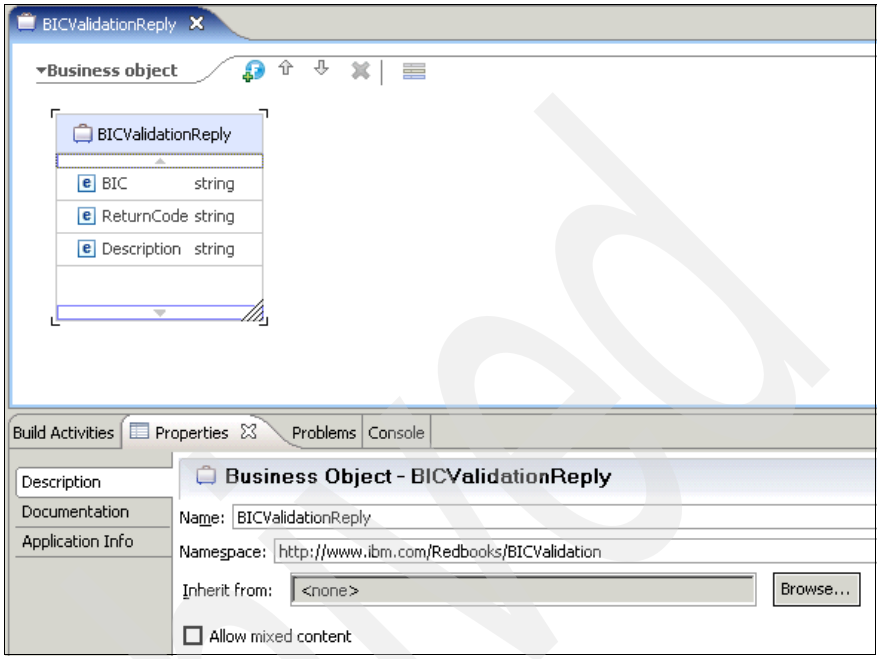

Figure 8-16 BICValidationReply business object

7. Create a new BICValidationService interface in the namespace `http://www.ibm.com/Redbooks/BICValidation` and give it a Request Response operation called `ValidateBIC`:

   a. Under Definitions, right-click **Interfaces** and select **New** → **Interface**.

   b. Enter the name and namespace, and then click **Finish**.

   c. Ensure that the new operation has one input parameter called `Request` of type `BICValidationRequest` and one output parameter called `Reply` of type `BICValidationReply`. After completing this step, the interface should look similar to the example in Figure 8-17.

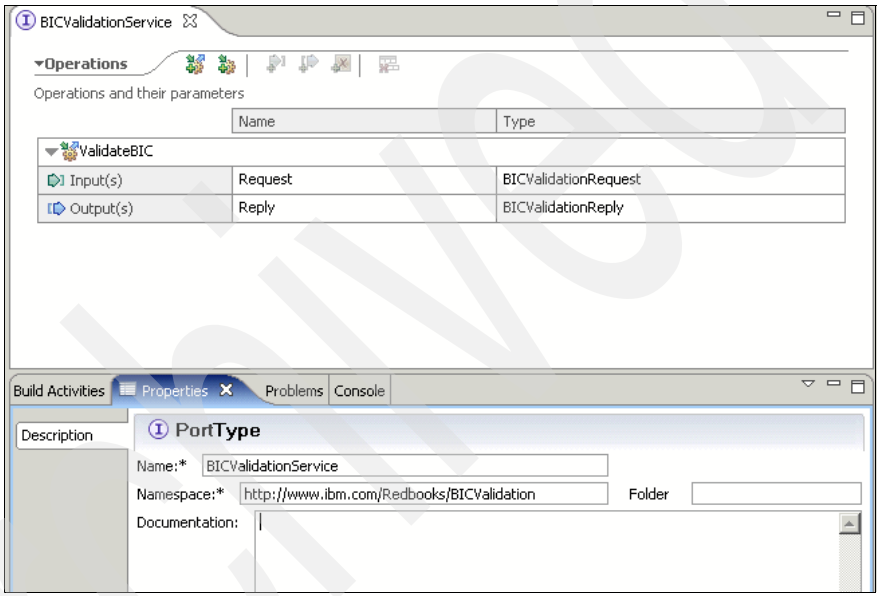

Figure 8-17 BICValidationService interface

   d. Save and close the interface.

8.3.3 Creating the WebSphere Transformation Extender maps

We now use the tooling to generate two WebSphere Transformation Extender maps:

- The first map is used when data comes in to the WebSphere ESB flow. This map converts the incoming data to the business object.
- The second map is used when data leaves the WebSphere ESB flow. This map converts the business object to the outgoing message.

Creating the WebSphere Transformation Extender project

To create the WebSphere Transformation Extender project:

1. Switch to the **Transformation Extender Development** perspective.
2. Create a new WebSphere Transformation Extender Project:
   a. Select **File** → **New** → **Extender Project**.
   b. In the New WebSphere Transformation Extender Project window (Figure 8-18), for Project name, type `WTX_Scenario_1_2`.

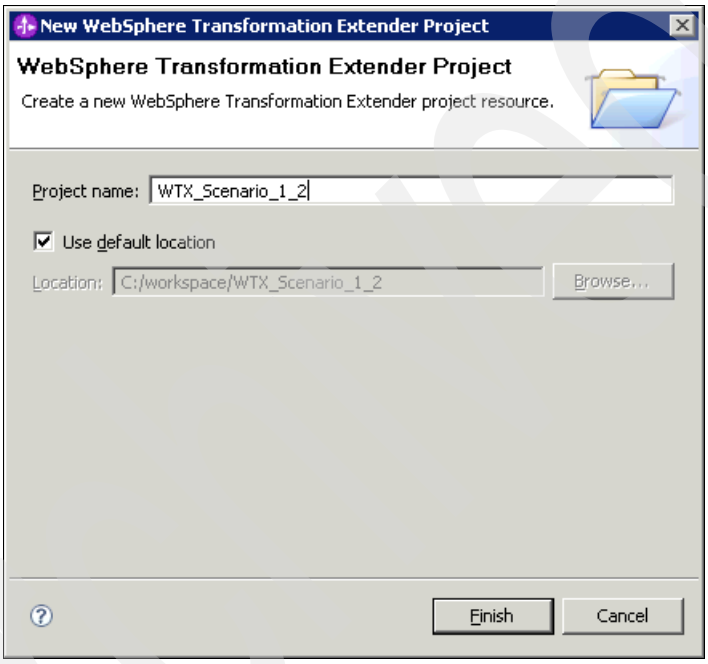

Figure 8-18 New WebSphere Transformation Extender Project window

Creating new artifacts by using the Map Generator for WebSphere ESB or WebSphere Process Server

To create new artifacts by using the Map Generator:

1. Create a new folder named `WTX` in the WTX_Scenario_1_2 project. Select **File** → **New** → **Other** and, from the General folder, select **Folder**.
2. Switch to the **Navigator** tab in Transformation Extender Development perspective.

3. Copy the **MT103Msg.xsd** schema file from the **Definitions** project to the **WTX** folder in the WTX_Scenario_1_2 project, as shown in Figure 8-19.

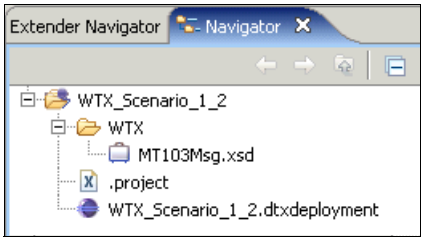

Figure 8-19 Navigator view

4. Switch back to the **Extender Navigator** tab in the Transformation Extender Development perspective.
5. Right-click the project folder and select **New** → **Map Generator for WESB/WPS** (Figure 8-20).

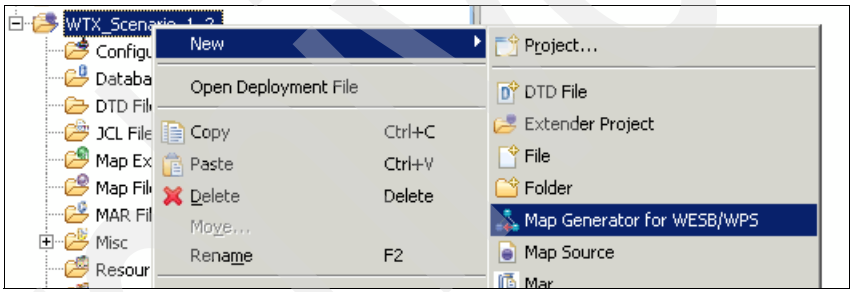

Figure 8-20 Running the Map Generator for WESB/WPS

6. In the Map Generator window (Figure 8-21), complete all the fields as follows:
   a. For BO Schema Location, click the ellipses (**...**) button and select the location of the `MT103Msg.xsd` file.
   b. For Native Content Type, type `SWIFT`.

> **Note:** The name for Native Content Type is used to construct the names of the generated maps. Remember the name because you must type the same name during creation of the data handler configuration. See “Creating the WTX data handler configuration” on page 532.

   c. For BO Type, verify that it is set to `MT103Msg`.
   d. Click **Next**.

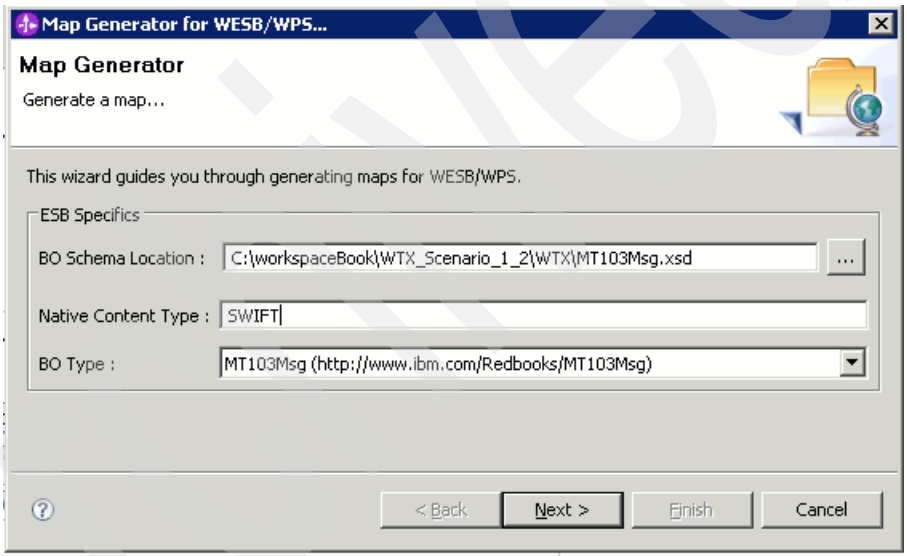

Figure 8-21 Map Generation for WESB/WPS window - ESB specifics

7. In the next window (Figure 8-22), for the destination folder of the generated files, select **WTX_Senario_1_2/WTX**. Verify that both **XML to Native map** and **Native to XML** are selected and click **Next**.

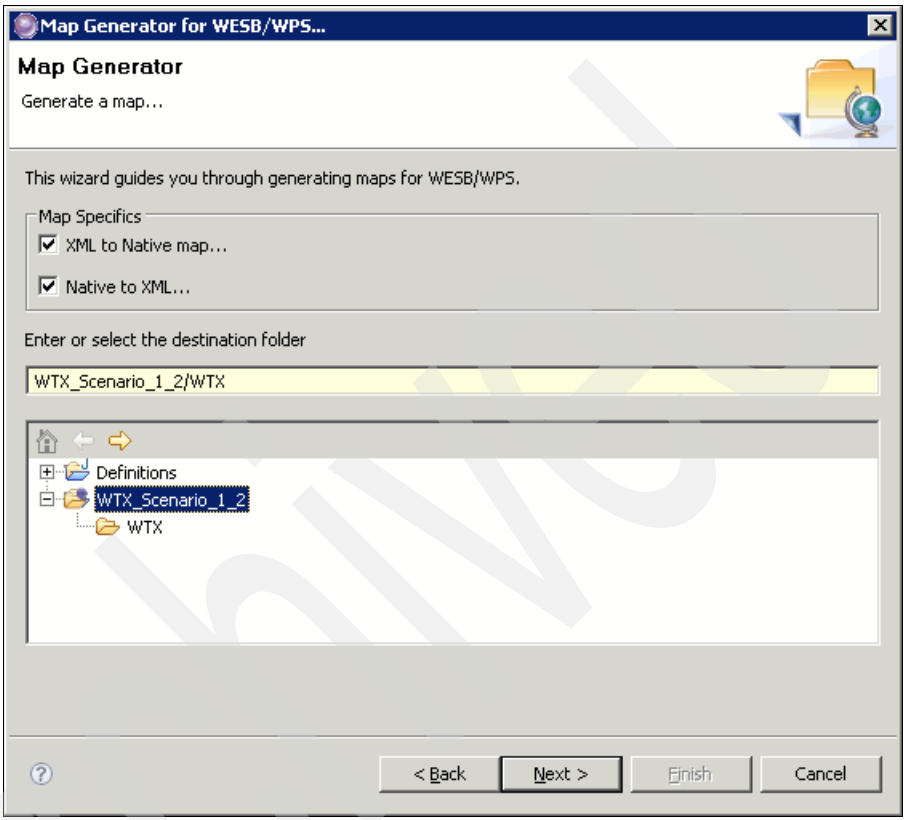

Figure 8-22 Map Generator for WESB/WPS window - Destination folder

8. Verify that everything completed successfully (Figure 8-23) and click **Finish**.

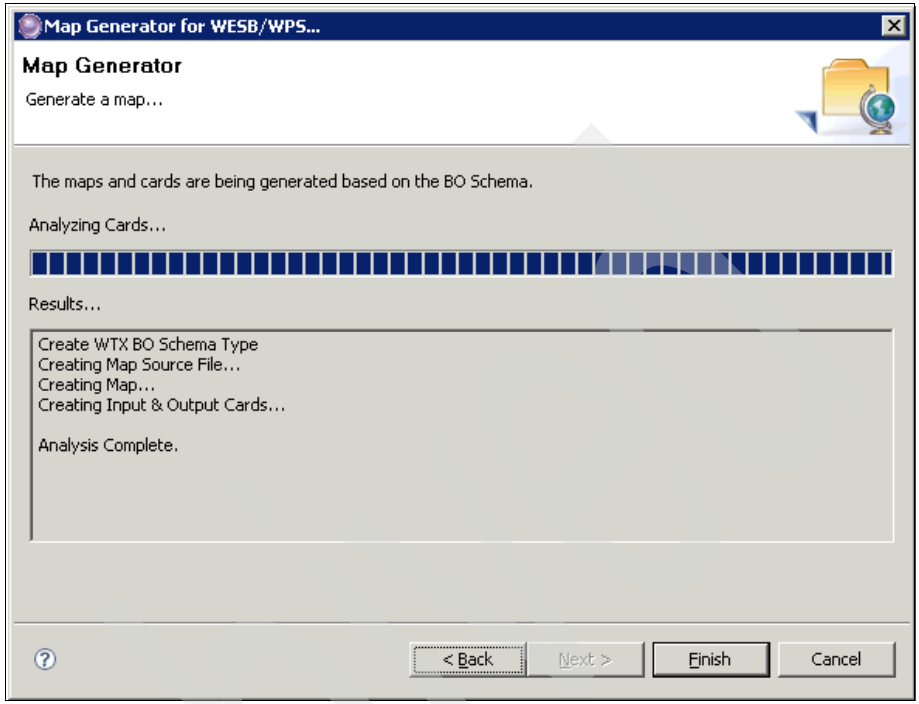

Figure 8-23 Map Generator for WESB/WPS window - Results

After completing these steps in your WTX_Scenario_1_2 project, you see the following new artifacts:

- The `MT103MsgSWIFT.mms` map source file

  This file contains two maps:

  – MT103MsgSWIFTToXML

    This map transforms the message from its native format to the business object. The tooling generated this map as having no input cards. We must add the input card for the SWIFT message. The tooling also generated this map with one output card that uses the MT103MsgXML format, which is used by the ESB.

  – MT103MsgXMLToSWIFT

    This map converts the message from the business object format to the native format. For this map, the tooling generated one input card (in the MT103MsgXML format that is used by the ESB). We must add the output card.

Note: When the WTX data binding or WTX data handler in WebSphere ESB or WebSphere Process Server uses these generated maps, it always uses the first input card and the first output card to pass the data in or out, or from or to, the WebSphere ESB or Process Server. You can also use other input/output cards in your map, but they will not be linked with WebSphere ESB or WebSphere Process Server. These cards use the adapters that you configure in the map in order to read and write data.

- The `MT103MsgElement.xsd` schema file, which is used as a type tree in transformation maps

  Figure 8-24 shows the content of this schema.

Note: The business object schema files cannot be used directly in Transformation Extender maps as a type tree, because they do not contain any elements. They only contain complex type definitions. This is why, during the execution of Map Generator for WESB/WPS, a new schema file is created (Figure 8-24). This new schema file has the same name as the original one with a suffix of Element. In our case, this is the `MT103MsgElement.xsd` file). The namespace of the new schema is created by adding `/wtx` to the original namespace (which in our case is `http://www.ibm.com/Redbooks/MT103Msg`**`/wtx`**).

In Figure 8-24, notice the element definition `<xs:element name="MT103Msg" type="bons:MT103Msg"/>`.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
targetNamespace="http://www.ibm.com/Redbooks/MT103Msg/wtx"
xmlns:bons="http://www.ibm.com/Redbooks/MT103Msg"
xmlns:wtx="http://www.ibm.com/Redbooks/MT103Msg/wtx">
<xs:import namespace="http://www.ibm.com/Redbooks/MT103Msg"
schemaLocation="MT103Msg.xsd"/>
<xs:element name="MT103Msg" type="bons:MT103Msg"/>
</xs:schema>
```

Figure 8-24 Contents of the MT103MsgElement.xsd schema file

- The `MT103MsgSWIFT.bcfg` file

  This is a binding resource configuration file that can be used to configure JMS, MQ and HTTP bindings.

Figure 8-25 shows all objects that are created during the execution of the Map Generator for WESB/WPS process.

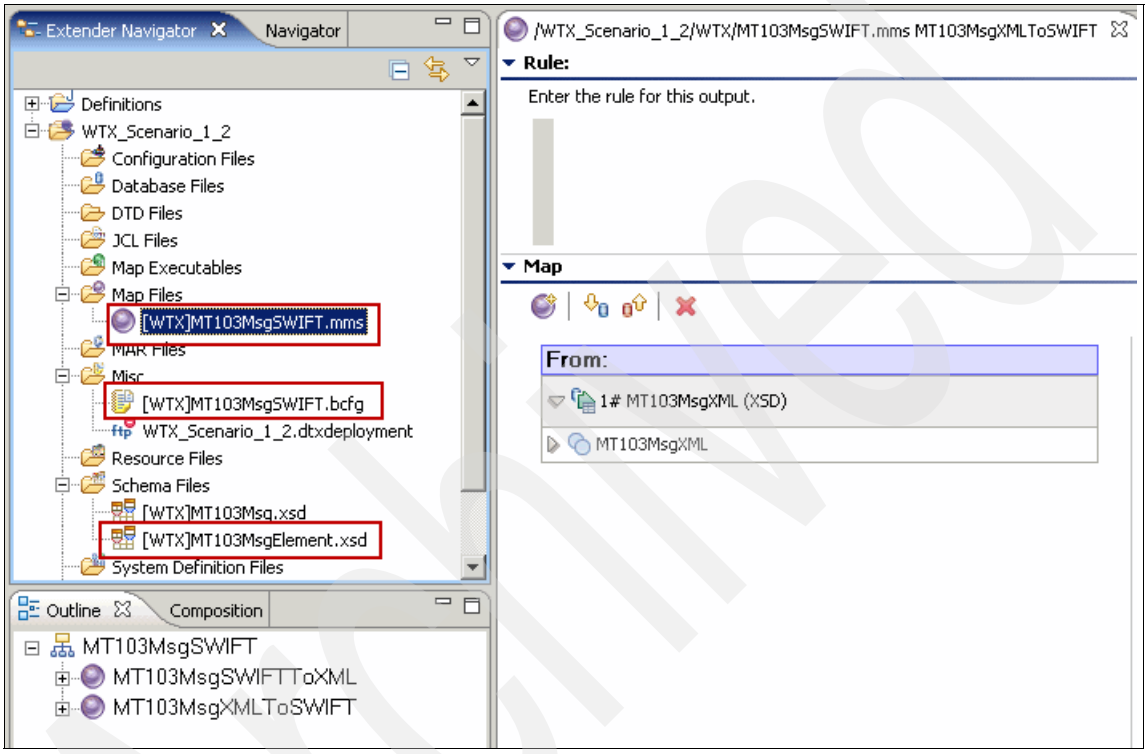

Figure 8-25 Objects created by running the Map Generator for WESB/WPS

8.4 Creating the first WebSphere Transformation Extender map

This first map transforms an incoming SWIFT MT103 message into an XML representation that is used by the ESB (Figure 8-26). This XML format is based upon the schema that is provided by the Map Generator for WESB/WPS. See “Creating new artifacts by using the Map Generator for WebSphere ESB or WebSphere Process Server” on page 497.

The map is a “one-to-one” map, because we have one input format (MT103) and one output format (ESB business object or WebSphere Message Broker message set) in a single map. This is the case for both of the generated maps.

Remember that the map generator has generated two maps. In each of the maps, it has added one card (either input or output) for the business object. Before we can map the individual fields, we must define the missing data structures (type trees) for each map. We start with the first generated map.

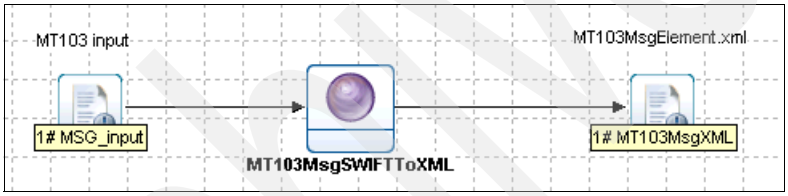

Figure 8-26 Overview of the MT103MsgSWIFTToXML map

8.4.1 Creating the type trees

We have predefined the type tree that we will use for the SWIFT MT103 message and explain how we constructed it.

Note: Normally we might use the SWIFTNet FIN industry pack that is provided by IBM. This pack contains a `swift_iso7775.mtt` type tree file that represents the MT103 data structure with the full validation built in. The type tree that we use is a strongly simplified version of this SWIFT type tree and is only suited for training purposes. To learn how to use the full SWIFTNet FIN type tree, see Appendix C, “Using type trees from the SWIFTNet FIN industry pack” on page 735.

You can import the predefined MT103.mtt type tree file into Design Studio:

1. Highlight your WTX Project and select **File** → **Import** → **General** → **File System**.
2. Navigate to the location where you have stored the MT103.mtt file and click **Finish**.

You can now navigate to the Type Trees folder and look at the type tree that we created.

A SWIFT MT103 message has many fields. If you use the SWIFTNet FIN pack, you have the complete type tree with all those definitions at your disposal. Because we built the type tree ourselves, we did not go through the effort of declaring all the possible fields in a SWIFT message. We only declared the relevant fields of the MT103 input structure. The relevant fields are those needed in the transformation process, based on fields of the output structure, that is the business object elements as shown in Example 8-2.

Example 8-2 Business object elements

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.ibm.com/Redbooks/MT103Msg">
   <xsd:complexType name="MT103Msg">
      <xsd:sequence>
         <xsd:element minOccurs="0" name="SenderBIC"
            type="xsd:string" />
         <xsd:element minOccurs="0" name="ReceiverBIC"
            type="xsd:string" />
         <xsd:element minOccurs="0" name="OrderingCustomer"
            type="xsd:string" />
         <xsd:element minOccurs="0" name="BeneficiaryCustomer"
            type="xsd:string" />
         <xsd:element minOccurs="0" name="Currency"
            type="xsd:string" />
         <xsd:element minOccurs="0" name="Amount" type="xsd:double" />
         <xsd:element minOccurs="0" name="AmountEUR"
            type="xsd:double" />
         <xsd:element minOccurs="1" name="InputMessage"
            type="xsd:string" />
         <xsd:element minOccurs="1" name="Status" type="xsd:string" />
         <xsd:element minOccurs="0" name="BICValidationReturnCode"
            type="xsd:string" />
```

```
        <xsd:element minOccurs="0" name="BICValidationReturnMsg"
           type="xsd:string" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:schema>
```

Therefore, we only needed to identify the fields listed in Table 8-2 in our input structure.

Table 8-2 Mapping table

Output fields	Mapping source
SenderBIC	The SenderReference from Application Header
ReceiverBIC	The ReceiverReference from Application Header
OrderingCustomer	The 50K segment from MT103
BeneficiaryCustomer	The 59a segment from MT103
Currency	The second field of 32A segment from MT103
Amount	The third field of 32A segment from MT103
AmountEUR	The result of the currency conversion web service
SwiftMessage	The whole MT103 input message
Status	The WTX validation result of the message (OK or SYNTAX ERROR)
BICValidationReturnCode	The result of the BIC validation web service
BICValidationReturnMsg	The result of the BIC validation web service

Figure 8-27 shows an MT103 example with some of the fields highlighted.

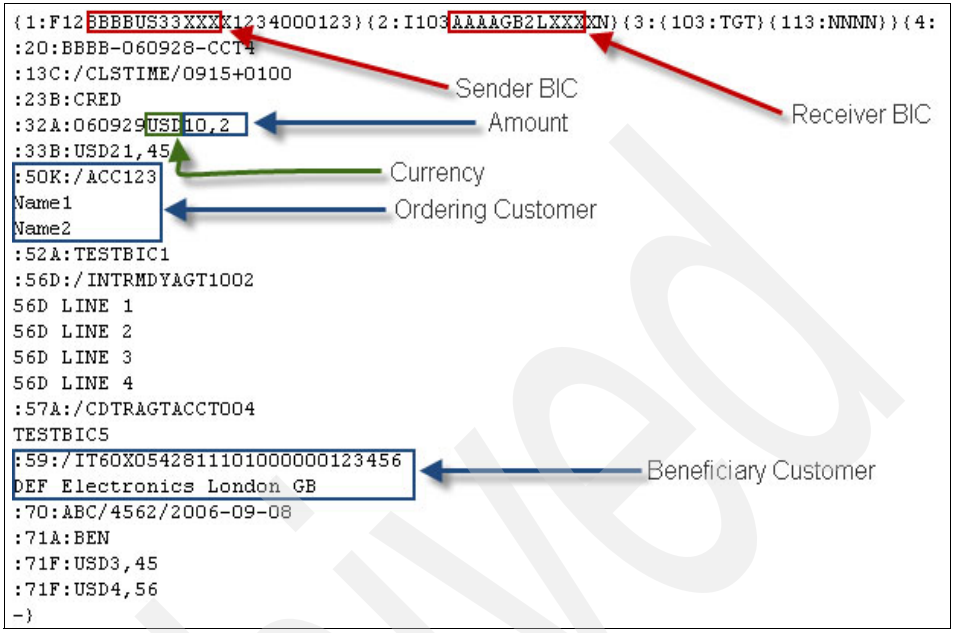

Figure 8-27 Example of the MT 103 SWIFT message and its relevant fields

The relevant fields of the MT103 input structure, highlighted in Figure 8-27, and shown in Table 8-2 on page 506 are items in our type tree. The top-level element is the InputMsg choice group, which is composed of an MT103 group and a BLOB, called *OtherMSG*. See 9.2.2, “Creating the type trees” on page 629, of the z/OS scenario, for more details about type tree creation.

The data stream should be split into MT103s and other messages. It is based on a component rule that checks whether the three leftmost characters match a standard SWIFT message application header initiation, not (`LEFT ($,3) = "{1:"`), and the properties of each data element.

Choice groups: Choice groups can be used to deal with a non-sequential data stream. Components might be items or groups, where the range can only be the default (1:1). The components of a choice group must also be distinguishable from each other. Data that is related to that group belongs exclusively to one of the components. In our case, any input file can be identified as an occurrence of either MT103 or OtherMSG.

Use restart attributes to avoid the execution return code 8 (“`one or more inputs was invalid`”) in case the input data does not meet MT103 validation criteria. You can see the component rule and the restart attributes in the Component View of the MT103 group, as shown at the bottom of Figure 8-28.

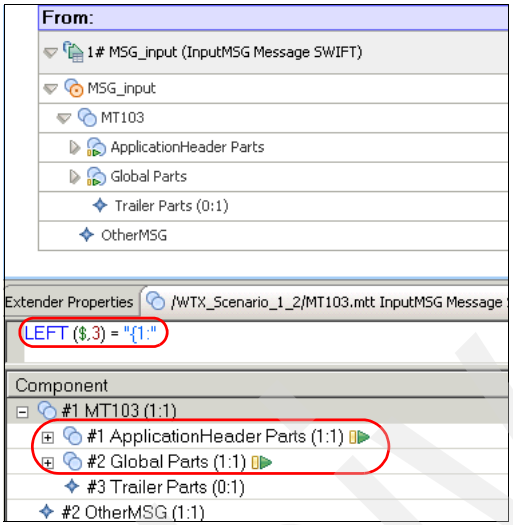

Figure 8-28 MSG_input group components and the MT103 component rule

We can distinguish our objects in the data stream by using the syntax properties. Keep in mind that any element that is declared as syntax property is not available as an element in the mapping. It also must be a fixed value (or set of fixed values).

For example, the first three characters of the MT103 basic header (which are{1:) are the syntax initiator of the SenderREF field. Figure 8-29 shows the SenderREF field in the Properties window. Also look at the syntax properties for the other data elements.

Property	Value
Name	SenderREF
Class	Item
Description	
Intent	General
⊞ Item Subclass	Text
Partitioned	No
Implied Default Value	
⊟ Type Syntax	
⊟ Initiator	Literal
Value	{1:
Ignore case	No
⊞ National Language	Western
⊞ Terminator	Literal
Release	None
Empty	None
Order subtypes	Ascending
⊟ Where Used	
As Component	ApplicationHeader Parts SWIFT

Figure 8-29 The SWIFT syntax format and syntax properties of our items

Table 8-2 on page 506 shows how we defined one item per relevant field.

We did not explicitly implement data fields for the data in the SWIFT message that we do not need in the map. Instead, we used BLOB items to define those data fields. An item cannot be reused in a sequential group. We can only tune the range of the consecutive occurrences. Therefore, we defined separate items for each BLOB of data that we want to define as shown in Figure 8-30 on page 510.

Notice in Figure 8-30 on page 510 that the name of the BLOBs have the word “details” followed by a sequential number, that is `details1` through `details4`.

In addition, look at the range of each object. We used the default range for each, except in the following cases:

- Details items

  These BLOBs should represent each occurrence of unrelevant data, terminated by <NL>. Here we used the (1:S) range.

- Name_address items

  These BLOBs can occur one to four times and are terminated by <NL>. Therefore, we used the (1:4) range.

- Trailer Parts item

  We decided to ignore the trailer in the validation process and, therefore, defined it as optional.

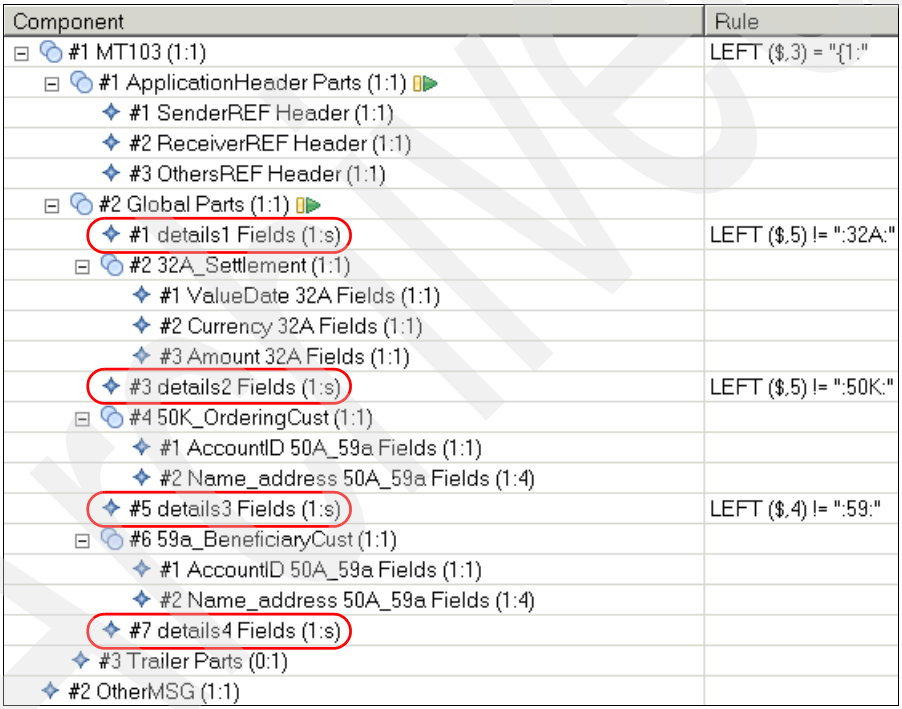

Figure 8-30 MT103 customized type tree structure

Before using any customized type tree (that is not generated by the type tree importer) in a transformation map, we must ensure that it is valid by creating a validation map.

8.4.2 Creating the validation map

We create a validation map that has the following characteristics:

- One input card, which is our SWIFT type tree, typed with the InputMSG choice group
- One output card with exactly the same type
- One mapping rule that copies all the data from the input to the output

To create a validation map for the type tree:

1. Create a new map.
2. Create an input card. In the Edit Input Card (Figure 8-31):
   a. For CardName, type a logical name. We chose `MSG_input`.
   b. Enter a type tree. We types `MT103.mtt`.
   c. Enter a type. We chose `InputMsg`.
   d. In the GET section, select an adapter. We use the default file adapter.
   e. Point to any valid data sample that the type tree should represent.

      You can use a test file in the compressed file (.zip) that you can download from the Redbooks site. See Appendix E, “Additional material” on page 749, for details.

      Notes:

      - You can point to a file in your project and to a file anywhere on your file system.
      - If you use the Microsoft Explorer to search for input data, the file adapter command line uses relative paths and back slashes. This map is not portable “as is” to another platform (especially on Linux or UNIX). Therefore, use the Resource Registry to make it portable. Alternatively, you can use a relative path with forward slashes (replace the back slashes with forward slashes manually). This is assuming that the relative path also exists on the other platform.

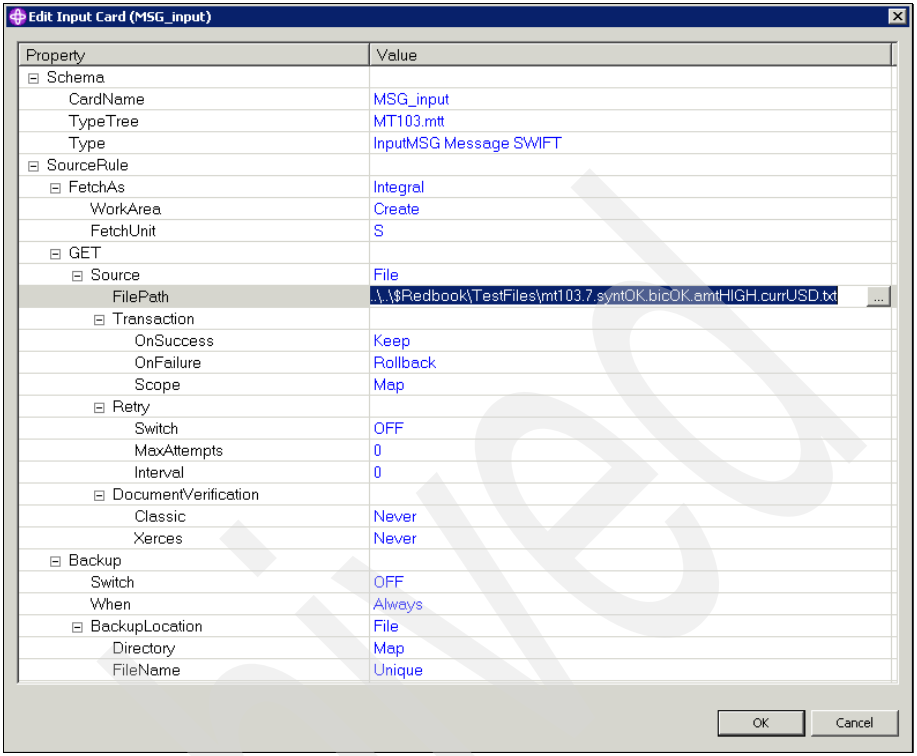

Figure 8-31 Input card definition - Schema and GET properties

3. Copy this input card and create an output card from it:
   a. Right-click the card level and select **Copy**.
   b. In the Copy Card window (Figure 8-32):
      i. Select on which map source and into which map you want to copy this card. The default is the same map in the same map source, so we leave this information unchanged.
      ii. Click **Output** to copy this card as an output card.
      iii. Change the name of the card.
      iv. Click **OK**.

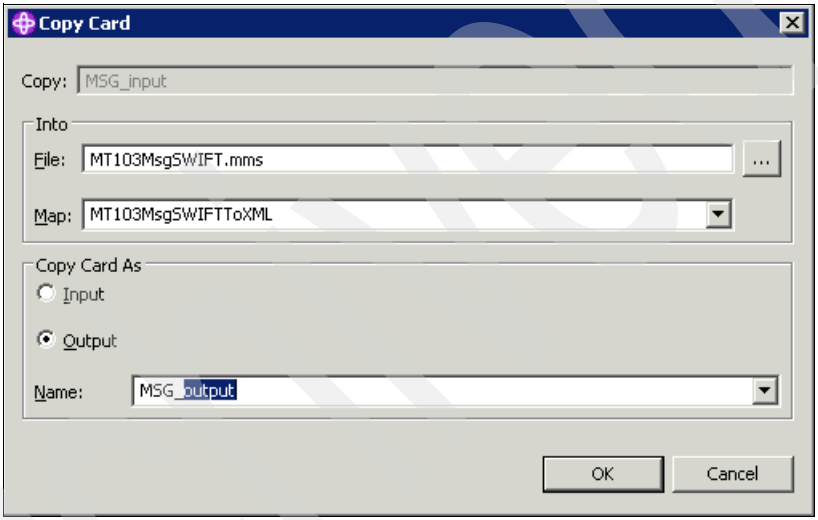

Figure 8-32 Copy Card window

4. Change the name of the output file in the output card that you just created. Otherwise, you will overwrite the input data file. Edit your new output card, and change the file name in the FilePath property.

Tip: Another fast and easy way to copy and paste a card is to use the outline view of the navigator and open the cards level. Right-click the appropriate input card and select **Copy**. Then right-click **Output Cards** and select Paste to copy the input card as an output card. The card is named `MSG_input_1`. Because each map card must have a unique name, rename that card to a proper logical name. This method also works across maps and even across map sources.

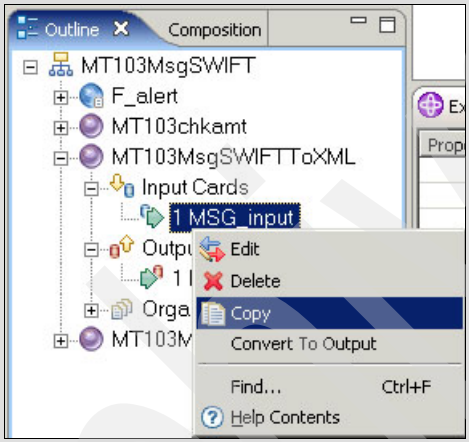

Figure 8-33 Quick copy and paste of a card using the navigator

5. Create the mapping rule for this validation map.

Tip: To map a validation map, always collapse the cards view to see only the top-level element of each card. This way you can only map the top level object input to the top level object output for the card configuration.

For any transformation map, always expand the cards view to show all data objects of our structures. By doing so, the output card rule column shows all white cells as mappable objects, where dark gray cells represent the objects that you cannot directly map at this stage (groups, multiple occurrence components, and so on). You can then collapse views for components that you want to map at the group level and create functional map calls if needed.

Drag the top-level object of the input card (the **MSG_input** choice group) to the top-level object of the output card (the **MSG_Output** choice group). A transformation rule (=Msg_Input) is created. This is the only mapping that is required for the validation map. Figure 8-34 shows how the map should look.

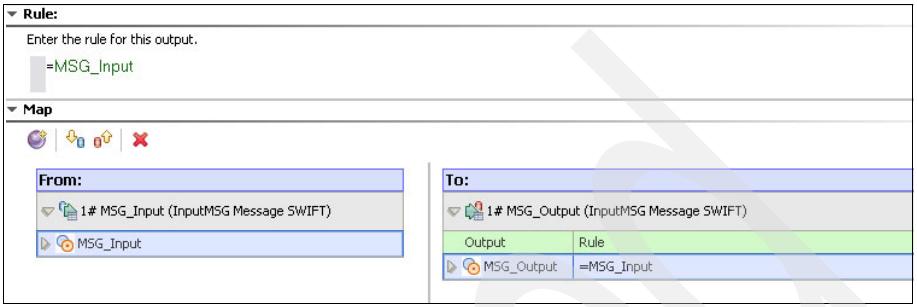

Figure 8-34 The validation map rule

6. Press Ctrl+B to build the map and then press Ctrl+R to run the map. The Command Server result window returns the "`map completed successfully`" message and as many output objects as input objects. This message indicates that your type tree works.

   If you see a message other than the success message, there is probably a mistake in the type tree definition. In this case, see Chapter 7, "Troubleshooting" on page 443 for assistance.

We can now develop the transformation map.

8.4.3 Creating the SWIFT to XML transformation map

To create the transformation map:

1. Select and copy the input card of the validation map.
2. Open the **MT103MsgSWIFTToXML** map that is generated by the Map Generator for WESB/WPS (in the `MT103MsgSWIFT.mms` map source).
3. Select the **MT103MsgSWIFTToXML** executable map, and paste the input card to this map.

Because the output card is the business object schema, the map looks similar to the example in Figure 8-35.

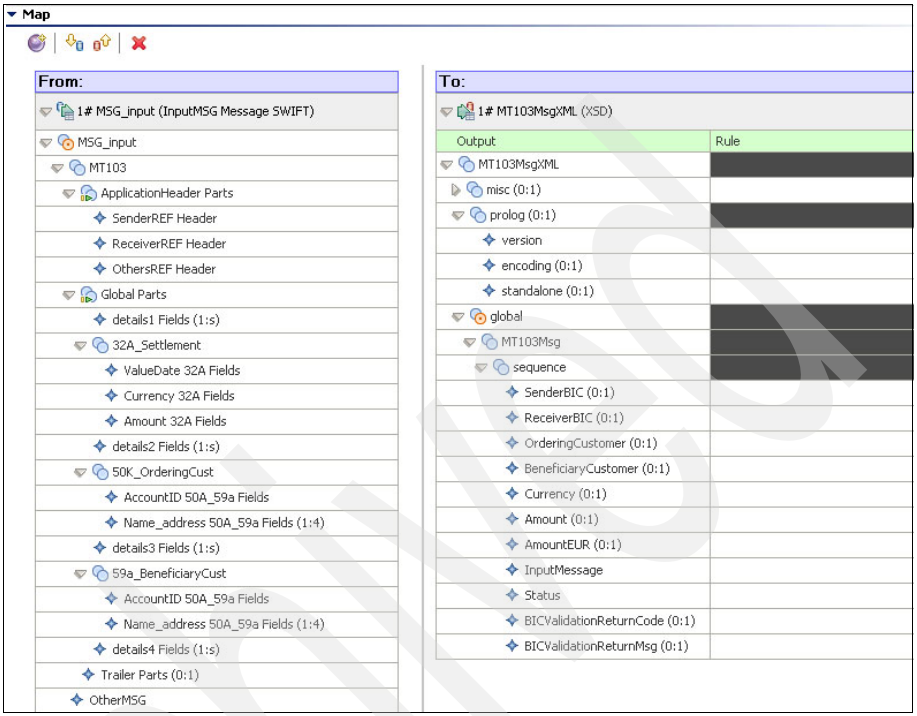

Figure 8-35 Getting ready for mapping MT103 to the business object

Tips: Before you begin mapping, keep in mind the following points:

- When you do the mapping, watch the object subclass (right-click and select **Show in properties**) and ensure that each mapping rule matches the object subclass. Use the subclass conversion functions (such as NUMBERTOTEXT) to transform to the correct subclass if needed.
- The optional objects do not need to be mapped. However, all objects must have a rule. Therefore, use the =NONE rule at group level or component level. Map the version and the encoding of any XML output.
- Drag fields from the source to the target and drag functions from the functions view, rather than typing their names. If you do not see the Functions window, select **Window** → **Show View** → **Functions**.

4. Map the prolog. Open the prolog group and map the Version field to "1.0" and the Encoding field to "UTF-8".
5. Continuing with the relevant data objects, encapsulate most of the rules in an EITHER function. This way, if the field is not present (meaning it is in error, because the fields are required on input), an error message can be produced. The two restart attributes (placed on ApplicationHeader and Global parts) allow the validation process to restart if any component of those groups is invalid. Use the following rules to map the fields:
   - SenderBIC

     Retrieve the sender's BIC by using an MID function that is applied on the SenderREF input item. The MID function has the following syntax:

     ```
     MID (source_text, position_to_start_the_search,
     number_of_characters)
     ```

     Figure 8-36 shows the rule.

     ```
     =EITHER (
     MID (SenderREF Header:ApplicationHeader Parts:MT103:MSG_input,7,11),
      "INVALID HEADER")
     ```

     Figure 8-36 Rule for the SenderBIC field

     Tips:

     - To force a mapping rule to continue on the next line to make it easier to read, press Ctrl+Enter.
     - You can also define senderREF as a group, composed of the six official fields of the SWIFT basic header (block, application and service identifiers, the LT address, the session number, and the sequence number). Then simply drag the LT address to retrieve the BIC. Leaving it as an item makes the validation process faster. However, in our case, performance is not an issue.

   - ReceiverBIC

     Retrieve the 11 leftmost positions of the input type ReceiverREF that has an initiator of {2:I103. Figure 8-37 shows the rule.

     ```
     =EITHER (
     LEFT (ReceiverREF Header:ApplicationHeader Parts:MT103:MSG_input,11),
      "INVALID HEADER")
     ```

     Figure 8-37 The rule for the ReceiverBIC field

– OrderingCustomer

Concatenate the account ID and the name_address of the ordering customer. Figure 8-38 shows the rule.

```
=EITHER (
TEXT (AccountID 50A_59a Fields:50K_OrderingCust:Global
Parts:MT103:MSG_input)+
TEXT (Name_address 50A_59a Fields:50K_OrderingCust:Global
Parts:MT103:MSG_input),
"INVALID MSG")
```

Figure 8-38 The rule for the OrderingCustomer field

– BenificiaryCustomer

Concatenate the account ID and the name_address of the beneficiary customer. Figure 8-39 shows the rule.

```
=EITHER (
TEXT (AccountID 50A_59a Fields:59a_BeneficiaryCust:Global
Parts:MT103:MSG_input) +
TEXT (Name_address 50A_59a Fields:59a_BeneficiaryCust:Global
Parts:MT103:MSG_input),
"INVALID MSG")
```

Figure 8-39 The rule for the BenificiaryCustomer field

– Currency

Copy this field from the input. Figure 8-40 shows the rule.

```
=EITHER (
Currency 32A Fields:32A_Settlement:Global Parts:MT103:MSG_input,
"INVALID MSG")
```

Figure 8-40 The rule for the Currency field

– Amount

Copy this from the input. Figure 8-41 shows the rule.

```
=EITHER (
Amount 32A Fields:32A_Settlement:Global Parts:MT103:MSG_input,
"INVALID MSG")
```

Figure 8-41 The rule for the Amount field

– InputMessage

Place the whole MT103 message here by using the PACKAGE function (includes all syntax objects). Figure 8-42 shows the rule.

```
=PACKAGE (MSG_input)
```

Figure 8-42 The rule for the InputMessage field

– Status

Testing whether you have a valid MT103 message. In this case, map it with “OK”. Otherwise this segment should show a “SYNTAX ERROR” message. Figure 8-43 shows the rule.

```
=IF (
PRESENT(REJECT (ApplicationHeader Parts:MT103:MSG_input)) |
PRESENT(REJECT (Global Parts:MT103:MSG_input)) |
PRESENT (OtherMSG:MSG_input),
"SYNTAX ERROR",
"OK")
```

Figure 8-43 The rule for the Status field

6. Test the presence of rejected data for the ApplicationHeader and Global group, which both have restart attributes (Figure 8-44). Alternatively, we test for the presence of another message, in which case there is a SYNTAX ERROR. Otherwise, the status is OK.

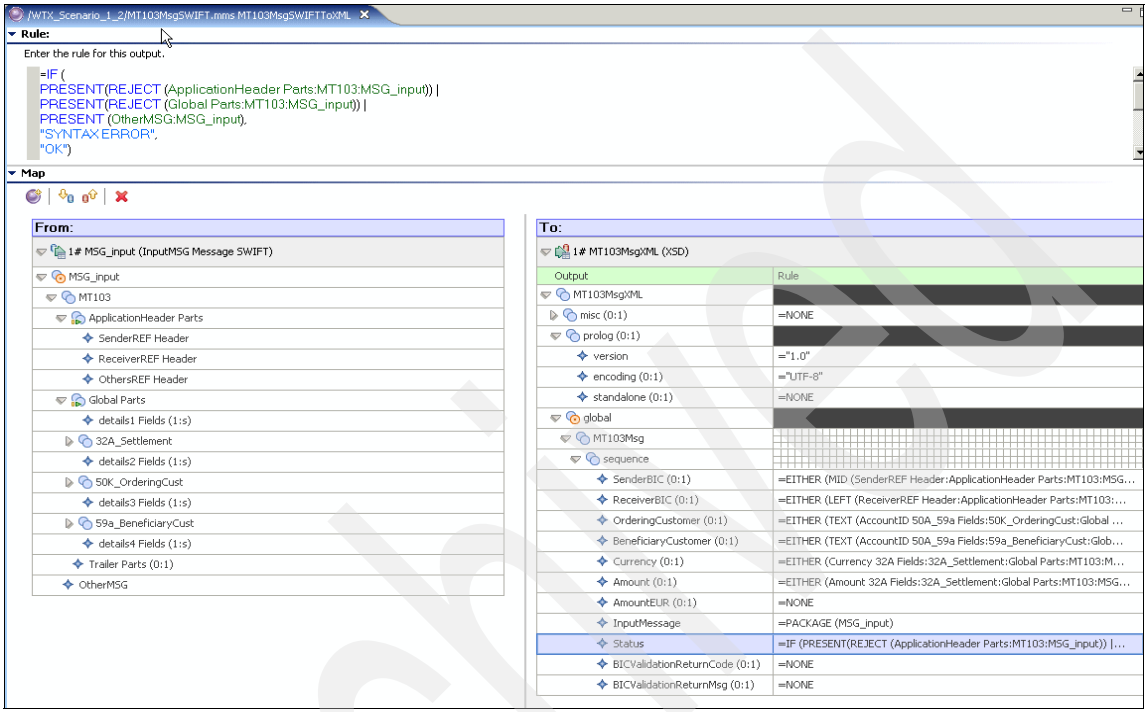

Figure 8-44 Mapping the Status output item

Pipe (|) symbol: The pipe symbolizes “or” logic, where “and” is “&” and different is “!=”.

Note: Again, this scenario has been simplified for demonstration purposes. You can use WebSphere Transformation Extender for a much more complicated and efficient validation process. Remember that the Industry Packs (SWIFT, EDI, and so on) provide deep and complete validation immediately upon use. For example, you can modify the validation by returning a specific status if an error is located in the ApplicationHeader or the Global groups.

You can then identify whether the error is in the header or in the global segment, by simply changing the rule for the Status output field as shown in Figure 8-45.

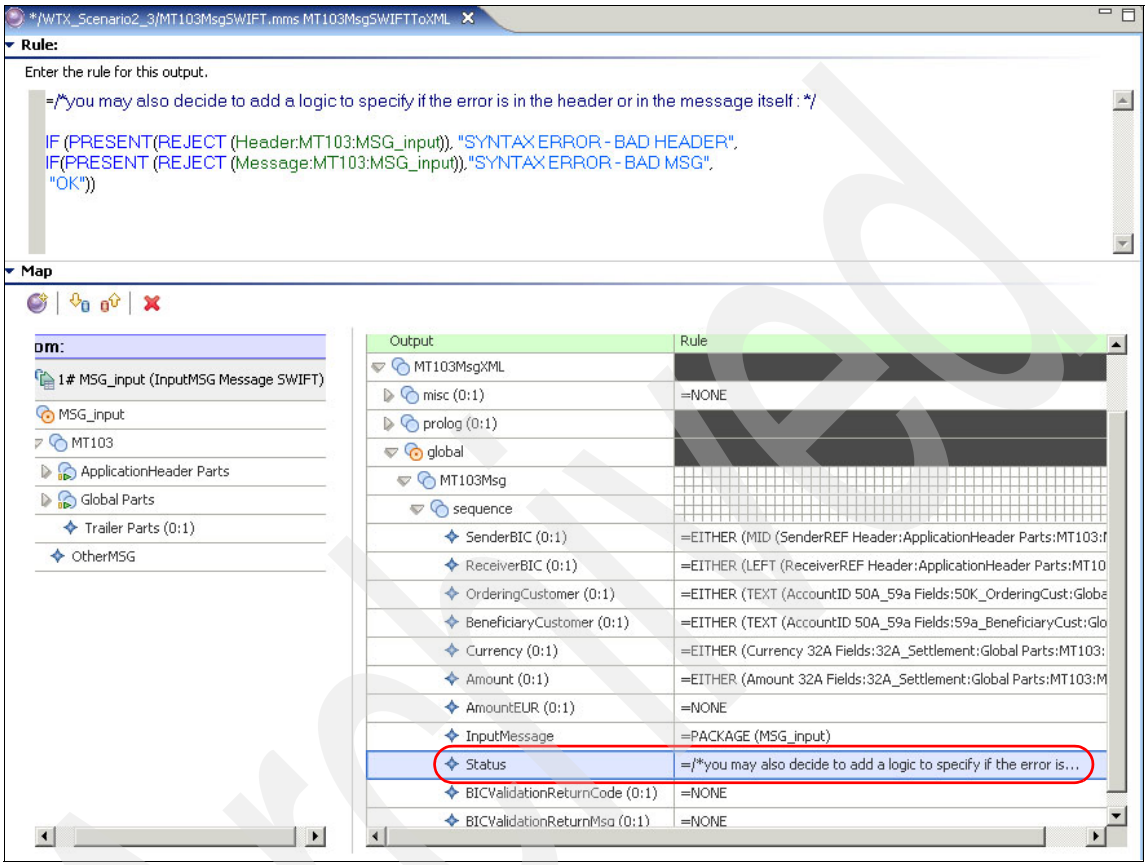

Figure 8-45 Modifying the mapping rule to tune the Status field result

Furthermore, you can consider simplifying the validation process by removing the two restart attributes from ApplicationHeader and Global groups, and put instead a restart on the “MT103” group. Therefore, the data is considered OtherMSG in case any of the MT103 group components are invalid because the validation restarts from the Global group in case of error and then consider the data stream as OtherMSG.

As shown in Figure 8-46, the mapping rule is as follows:

```
=IF (
PRESENT(REJECT (MT103:MSG_input)) |
PRESENT (OtherMSG:MSG_input),
"SYNTAX ERROR",
"OK"))
```

The AmountEUR, BICValidationReturnCode and BICValidationReturnMsg output items are populated by using Web services calls. Similar to all the other optional objects, these fields can be mapped with the =NONE rule. Right-click and select **Insert NONE if Empty** to map every empty rule column with =NONE). This concludes the first map.

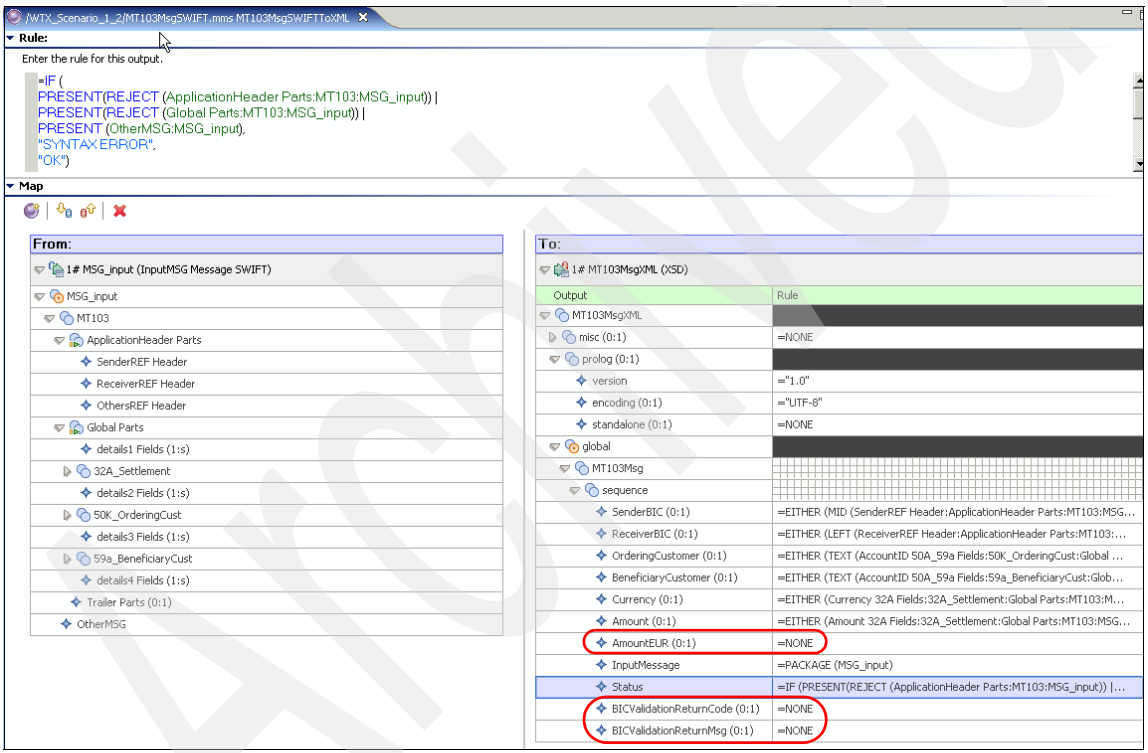

Figure 8-46 Mapping MT103 to the XML business object

7. Test the map. Edit the output card to change the Target to **File** and configure a FilePath for the output file.

   Attention: The card properties will be overridden by the integration server (WebSphere ESB or WebSphere Message Broker). However, for unit testing the map, we use the File adapter.

8. Build and run the map by using all possible test files in the TestFiles folder. Every message should allow the map to complete with a return code of 0 (“`map completed successfully`”). You should always get an XML structure on output.

8.4.4 Creating the XML to SWIFT executable map

Next, we complete the MT103MsgXMLToSWIFT executable map (Figure 8-47) for WebSphere ESB to use.

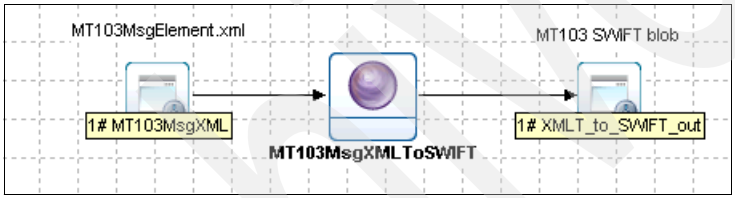

Figure 8-47 Overview of the MT103MsgXMLToSWIFT map

By using the MT103MsgXMLToSWIFT map, we can route the original MT103 message, extracted from the <InputMessage> element of the business object, to various destinations. (See “The process for scenario 1: WebSphere ESB” on page 477 for details.) Again, this is a one-to-one executable map, with one input card using the business object as native XSD format, and one output card extracting the original MT103 from the business object into a BLOB. We can use the BLOB definition of the OtherMSG type from the `MT103.mtt` file to do so.

To complete this map:

1. Look at the generated MT103MsgXMLToSWIFT map and verify that the input card is already created, but no output card exists yet.
2. Add an output card to the MT103MsgXMLToSWIFT executable map.

3. Use the `MT103.mtt` file as type tree file and select the **OtherMSG** type when defining the schema of the card. Figure 8-48 shows how your settings should look.

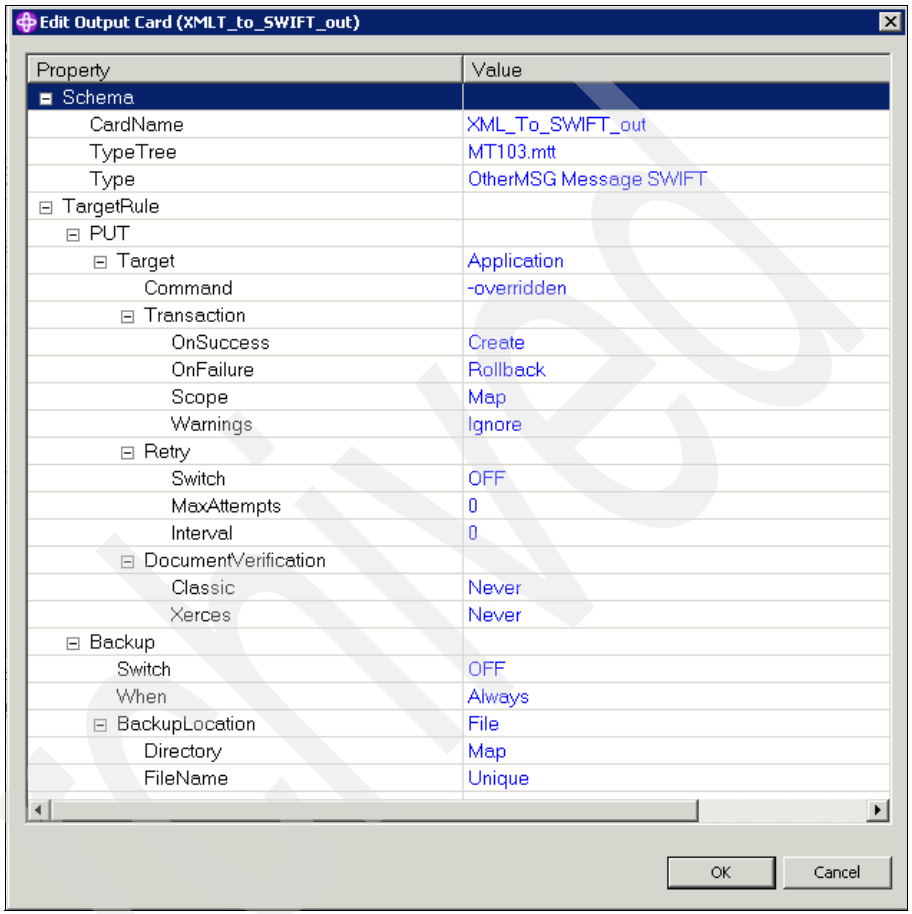

Figure 8-48 Output card of the MT103MsgSWIFTToXML map

4. Add the name of an output file and set the Target adapter to **File**, so you can unit test the map. Remember that this output file setting will be overridden later by WebSphere ESB.
5. Edit the input card and change the Get → Source adapter to **File** and use one of the output files of the MT103MsgSWIFTToXML map as your input (from 8.4.3, “Creating the SWIFT to XML transformation map” on page 515).

6. Map the **InputMessage** item to the **OtherMSG** output item, encapsulated in the TEXT function.

> **TEXT function:** The TEXT function returns the content of an object, excluding the initiator and terminator of that object.

Figure 8-49 shows the rule.

```
=TEXT(InputMessage:sequence:MT103Msg:global:MT103MsgXML)
```

Figure 8-49 The rule for the XML_To_SWIFT_out field

Figure 8-50 shows both the rule and the details of mapping the InputMessage item to the OtherMSG item by using a TEXT function.

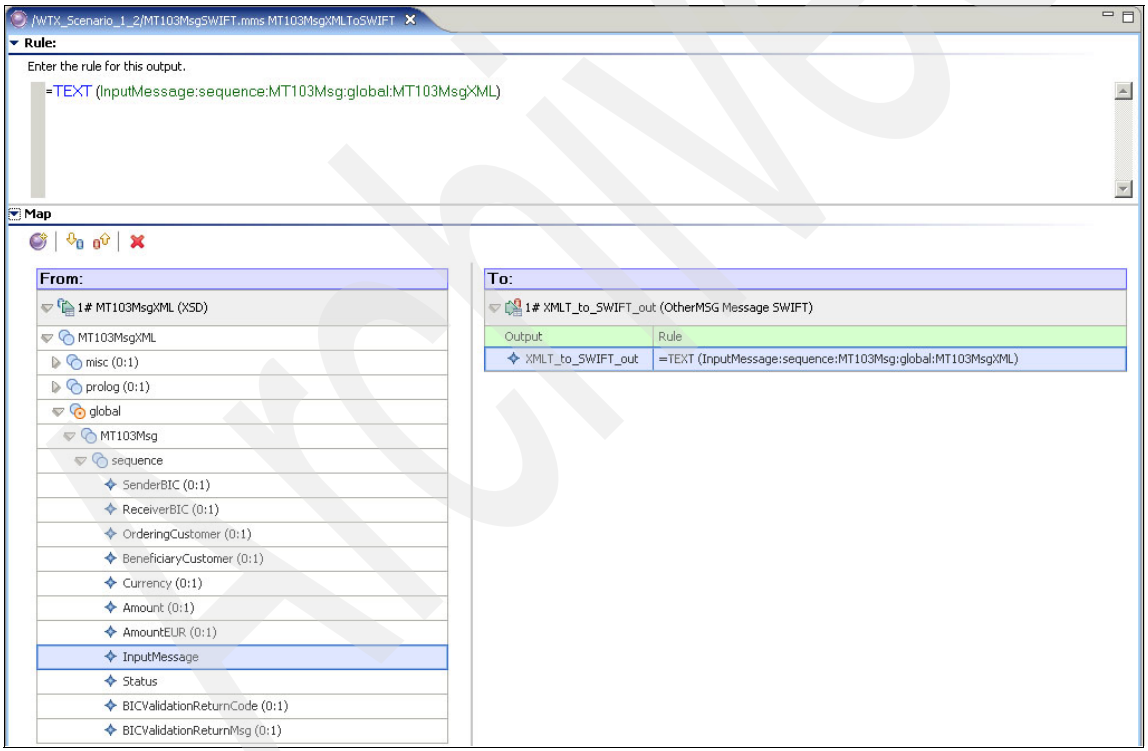

Figure 8-50 Mapping the InputMessage item to the OtherMSG item by using a TEXT function

7. Build and run the map. You should see that this map (MT103MsgXMLToSWIFT) extracts a valid MT103 message from the input file.

8.4.5 Building the WebSphere ESB scenario

You now have two maps that transform the MT103 message from its native SWIFT format to the business object and vice versa. In the following steps, we explain how to create a simple application that transforms a file from a specific folder to a business object by using the WTX MapSelection data handler called by the WebSphere Adapter for Flat Files.

Copying the compiled maps to the Definitions library

To copy the compiled maps to the Definitions library:

1. Switch to the Business Integration perspective in WebSphere Integration Developer.
2. In the Definitions library, created in 8.3.2, “Preparing the artifacts” on page 491, create a new folder named `WTX`.
3. In the Business Integration perspective, click the **Physical Resources** tab.
4. Copy the compiled maps from the **WTX_Scenario_1_2/WTX** folder to the **WTX** folder (created in step 2) as shown in Figure 8-51.

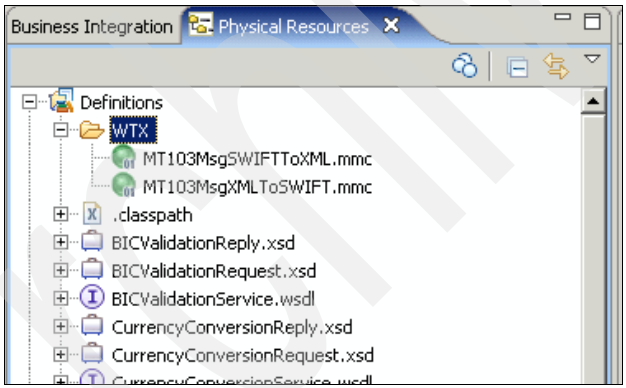

Figure 8-51 Copy the compiled maps to the Definitions library

5. Copy the `MT103MsgElement.xsd` file from the **WTX scenario_1_2** project to the **Definitions** project.

Creating a new mediation module

To create a new mediation module:

1. Switch back to the **Business Integration** tab in the Business Integration perspective.
2. Create a new mediation module:
   a. Select **File** → **New** → **Mediation Module**.
   b. In the New Mediation Module - Mediation module window (Figure 8-52):
      i. For Module name, enter the name `ESB_Scenario1`.
      ii. For Target runtime, choose the appropriate server.
      iii. Verify that the **Create mediation component** option is selected.
      iv. Click **Next**.

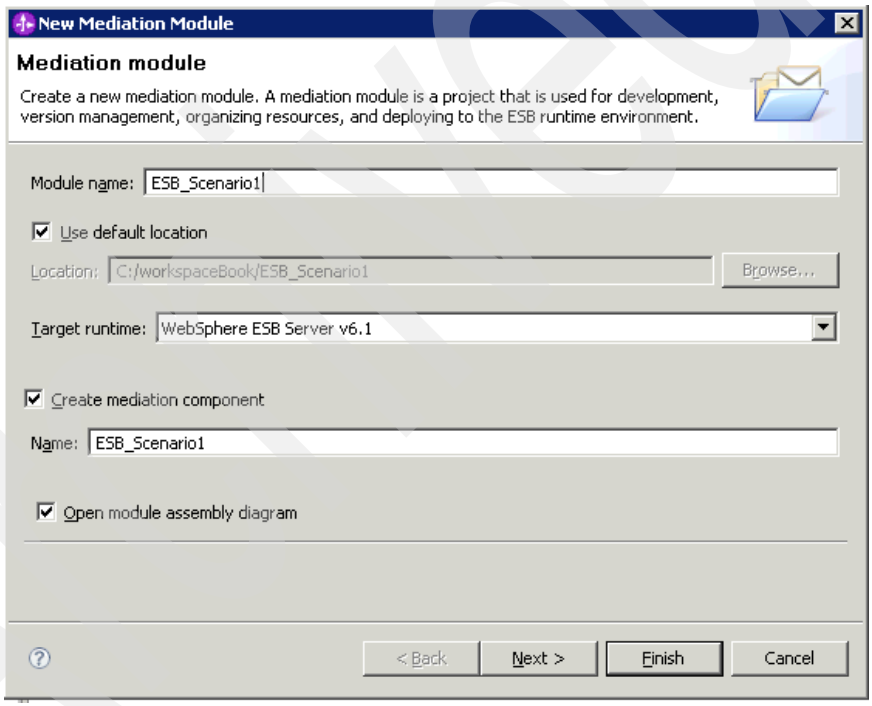

Figure 8-52 New Mediation Module window - Creating a mediation module

c. In the New Mediation Module - Select required libraries window (Figure 8-53), select **Definitions** to add the library to the dependencies. Then click **Finish**.

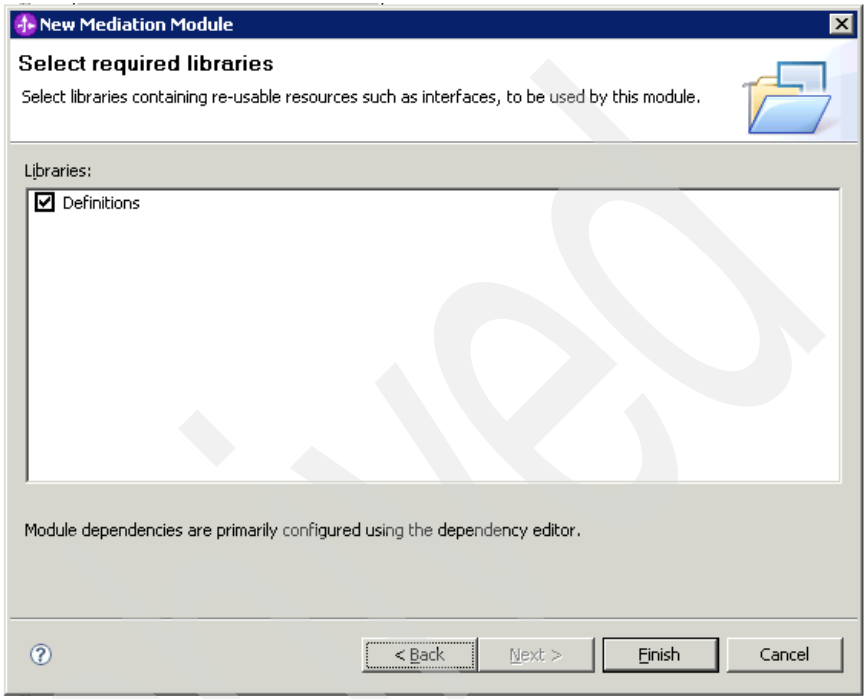

Figure 8-53 New Mediation Module window - Selecting the required libraries

Importing the adapter RAR file

To import the RAR file for WebSphere Adapter for Flat Files:

1. In the **Business Integration** view, right-click the **ESB_Scenario1** project and select **Import**.
2. In the Import - Select window (Figure 8-54), from the list of possible imports, in the J2EE folder, select **RAR file**.

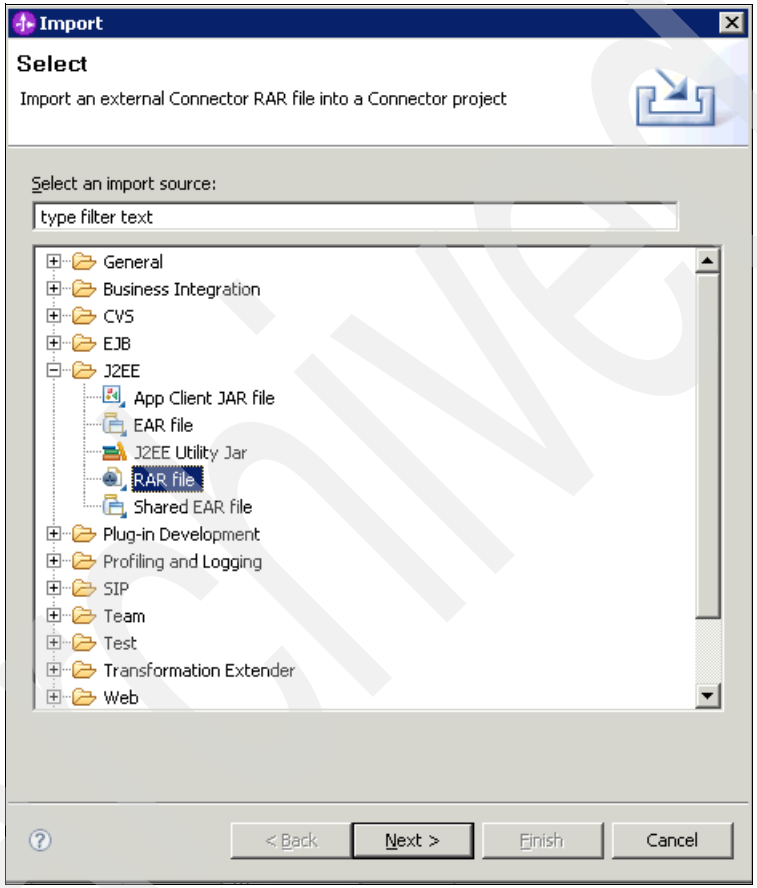

Figure 8-54 Importing the Flat File adapter

3. Import the FlatFile connector module from the file system. The `CWYFF_FlatFile.rar` file for Flat File adapter is in your WebSphere Integration Developer `\ResourceAdapters\FlatFile_6.1.0.0_IF04\deploy` installation folder.

4. In the Import - Connector Import window (Figure 8-55), specify the following settings:
   a. For Connector file, select the `CWYFF_FlatFile.rar` file in the `\ResourceAdapters\FlatFile_6.1.0.0_IF04\deploy` folder.
   b. For Connector module, enter `CWYFF_FlatFile`.
   c. Select the **Add project to an EAR** check box.
   d. For EAR Project Name, select **ESB_Scenario1App**.
   e. Click **Finish**.

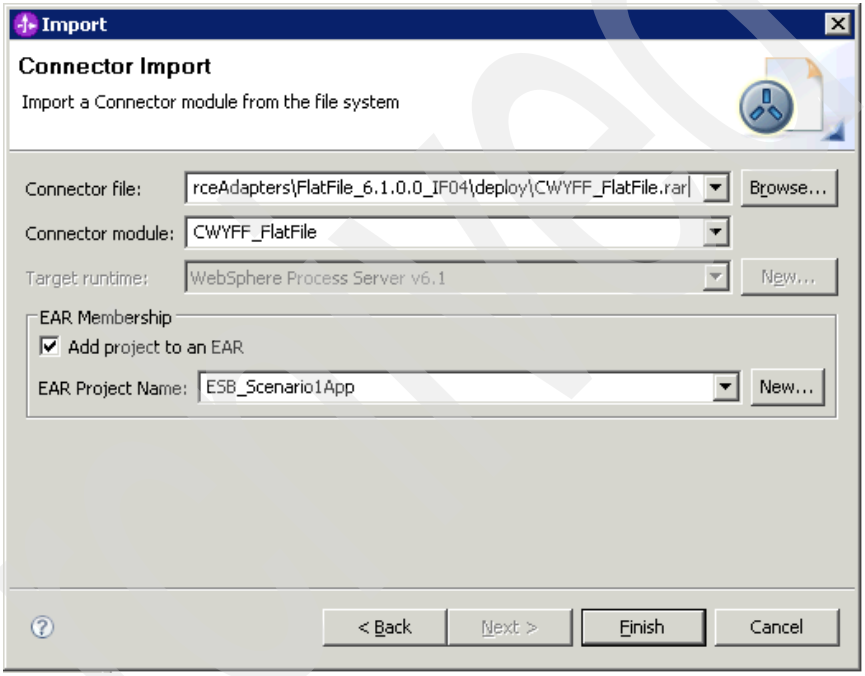

Figure 8-55 Importing the Flat File adapter

5. In the next window, click **No** if prompted to open the J2EE perspective.
6. Right-click the **ESB_Scenario1** project and select **Open Dependencies**.

7. On the Business Dependencies page (Figure 8-56), expand the **J2EE (Web, EJB, Connector, etc)** section and add **CWYFF_FlatFile** to your module.

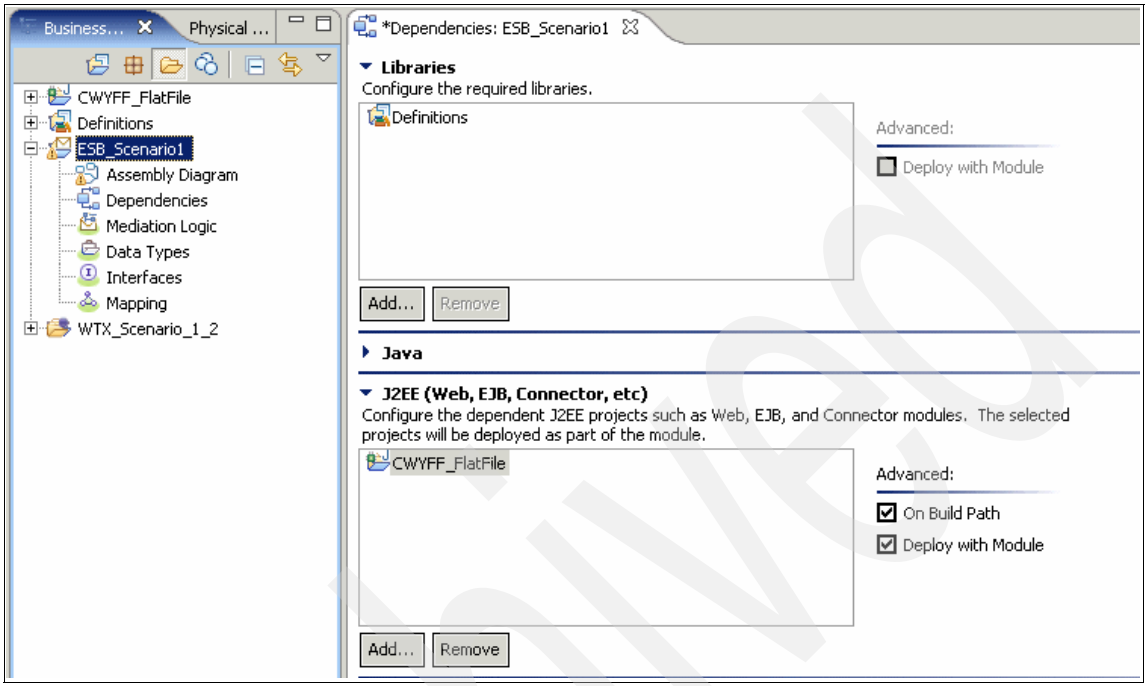

Figure 8-56 Adding the Flat File adapter to projects dependencies

8. Save and close the Dependencies window.

Creating the binding resources

In the next two sections, we create the binding resources so that WebSphere ESB can use WebSphere Transformation Extender to do the transformations. First, we create a configuration for the *data handler* so that we can specify which maps to use. Then, we create a configuration for the *data binding*. With this configuration, we can specify where the data is retrieved and sent (files) and how they are transformed to and from business objects (using the new data handler).

Figure 8-57 shows the artifacts that we create in the next two sections, what they specify, and how they relate to each other.

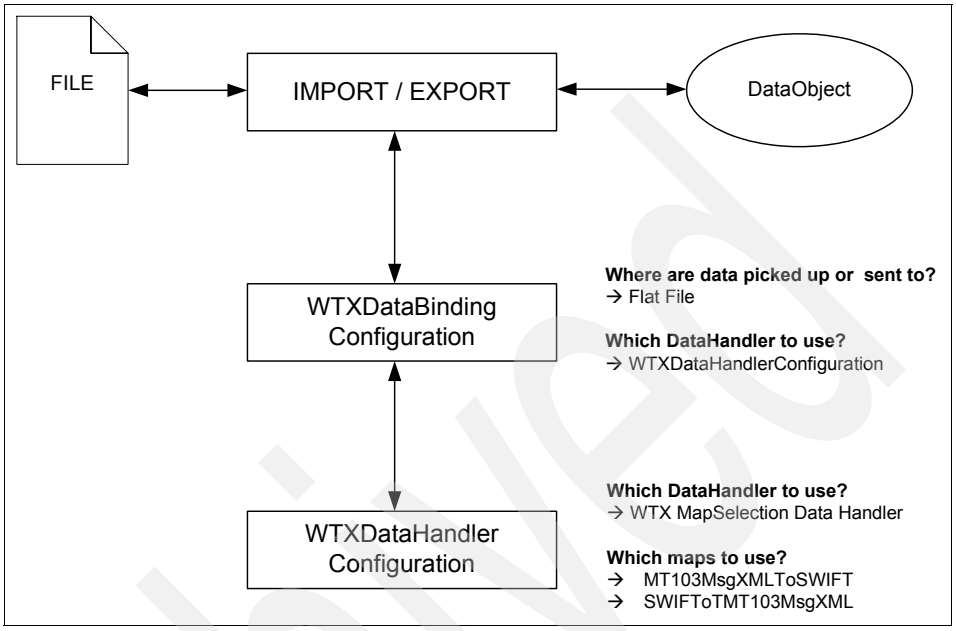

Figure 8-57 The WTX data binding and data handler

Creating the WTX data handler configuration

In this section, we create a data handler resource binding configuration. The WebSphere Adapter for Flat Files provides a data binding class that can be extended by data handlers. In our case, we use the WTX MapSelection data handler class.

1. Right-click the **ESB_Scenario1** module and select **New** → **Binding Resource Configuration** (Figure 8-58).

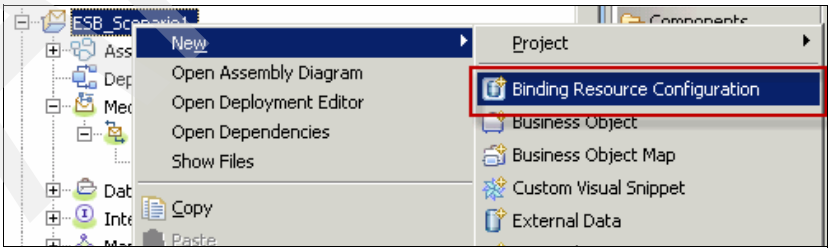

Figure 8-58 Selecting Binding Resource Configuration

2. In the Binding Resource Configuration window (Figure 8-59), for Name, type `WTXDataHandlerConfiguration`, and for Namespace, type `http://www.ibm.com/Redbooks/SWIFTMessagePreprocessing`. Click **Next**.

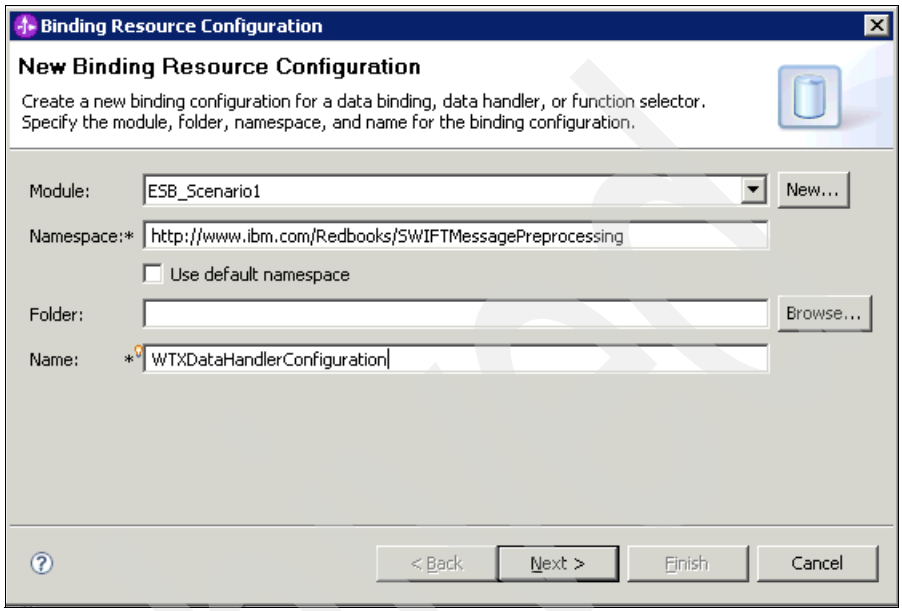

Figure 8-59 Binding Resource Configuration window

3. In the next Binding Resource Configuration:
   a. For Configuration Type, select **Data Handler**.
   b. For data handler class name, click **Browse**.
   c. In the Data Handler Selection window (Figure 8-60), from the Matching data handlers list, select **WTX MapSelection Data Handler** and click **OK**.

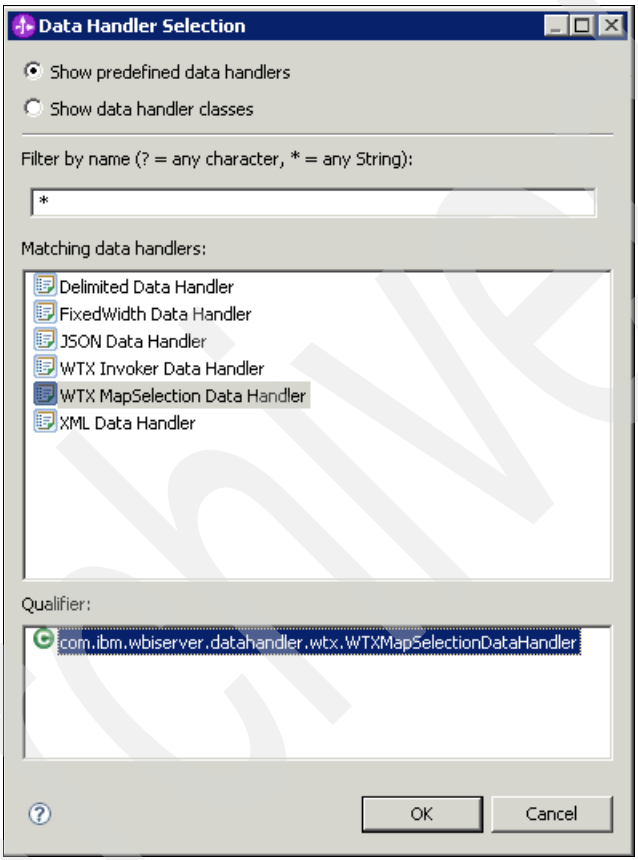

Figure 8-60 Data Handler Selection window

d. Back in the Binding Resource Configuration - Select a Configuration Type window (Figure 8-61), verify that the class name is set to `com.ibm.wbiserver.datahandler.wtx.WTXMapSelectionDataHandler` and click **Next**.

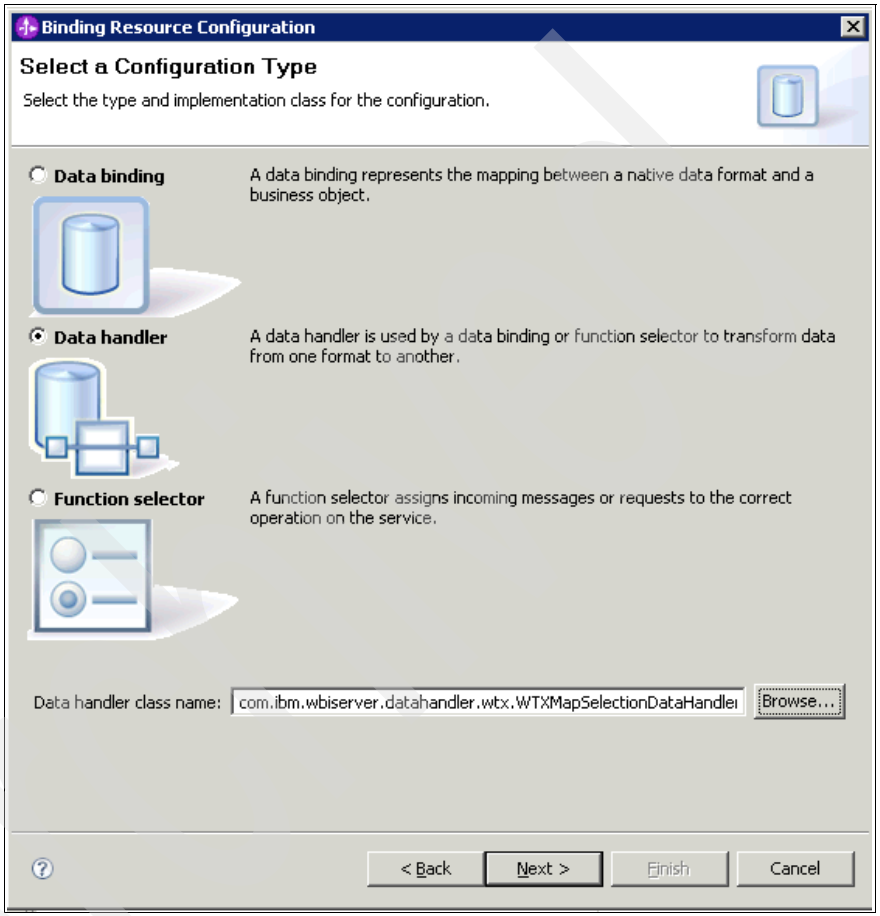

Figure 8-61 Binding Resource Configuration wizard

4. In the Binding Resource Configuration - Data Handler Properties window (Figure 8-62), for Content Type, type `SWIFT`.

> **Attention:** Use the same name that you used during the execution of the Map Generator for WESB/WPS in step 6 on page 499.

For TransformToNative Map Name and TransformToXMLMap Name, the `$(ContentType)` is resolved at run time to a given name. Based on the provided value and business object name, the map name is created. In our case, the business object is named `MT103Msg`. Therefore, the template `$(BusinessObject)XMLTo$(ContentName)` resolves to `MT103MsgXMLToSWIFT`, which is the same name that the Map Generator created.

Click **Finish**.

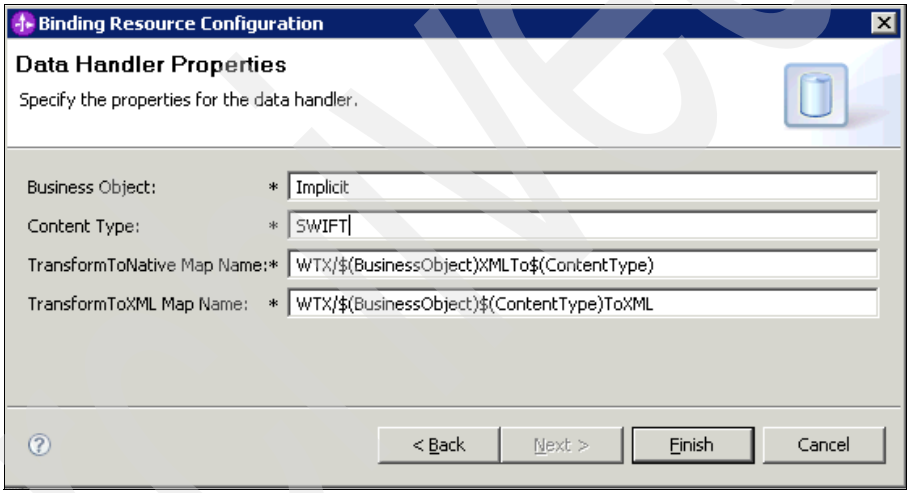

Figure 8-62 Binding Resource Configuration

5. Close the WTX data handler configuration.

Creating the WTX data binding configuration

We create a new data binding resource configuration that will use the FlatFileBaseDataBinding class. We configure the FlatFileBaseDataBinding class to use a WTX data handler to transform the data.

> **Note:** In 8.3.3, “Creating the WebSphere Transformation Extender maps” on page 496, we use the Map Generator for WESB/WPS. One of the results of using this tool is a resource binding configuration file. This file uses the WTXDataBinding class and cannot be used with resource adapters.

1. Right-click the **ESB_Scenario1** module and select **New** → **Binding Resource Configuration** (Figure 8-58 on page 532).
2. In the Binding Resource Configuration - New Binding Resource Configuration window (Figure 8-63), for Name, type `WTXDataBindingConfiguration`, and for Namespace, type `http://www.ibm.com/Rebooks/SWIFTMessagePreprocesing`. Then click **Next**.

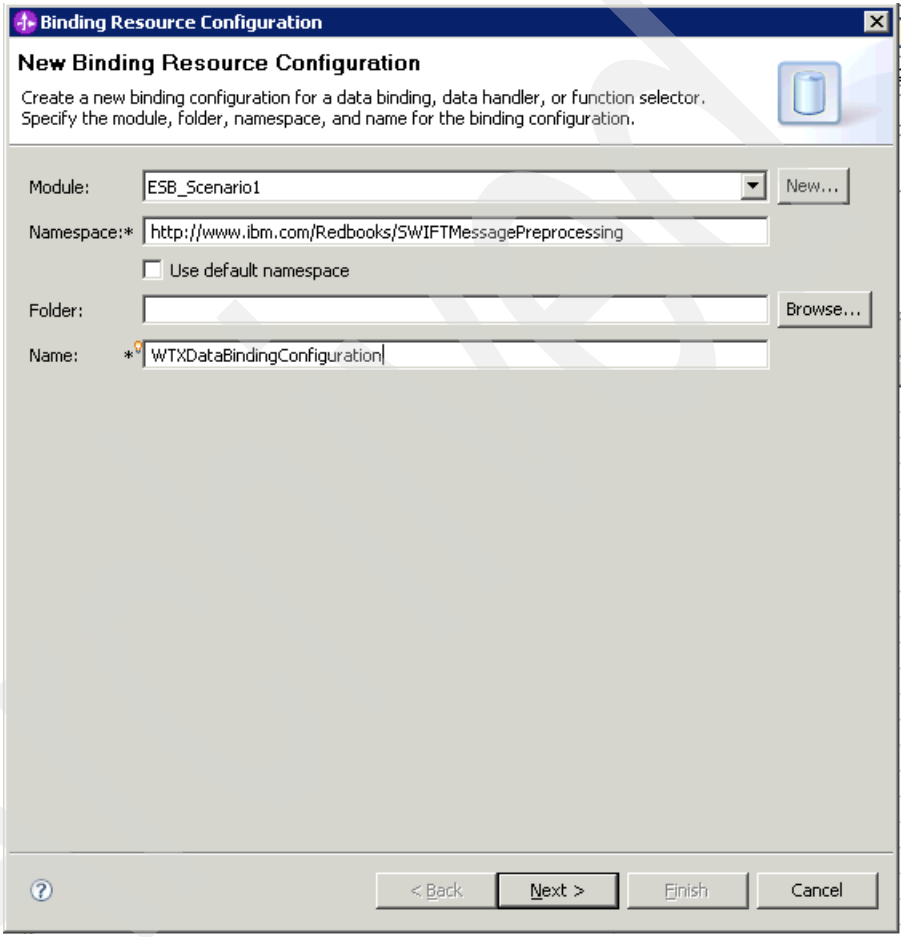

Figure 8-63 Binding Resource Configuration - New Binding Resource Configuration window

3. In the Binding Resource Configuration - Select a Configuration Type window:
   a. For the Configuration Type, select **Data binding.**
   b. For Data binding class name, click the **Browse** button.
   c. In the Data Binding Selection window (Figure 8-64), select **FlatFileBaseDataBinding** and click **OK**.

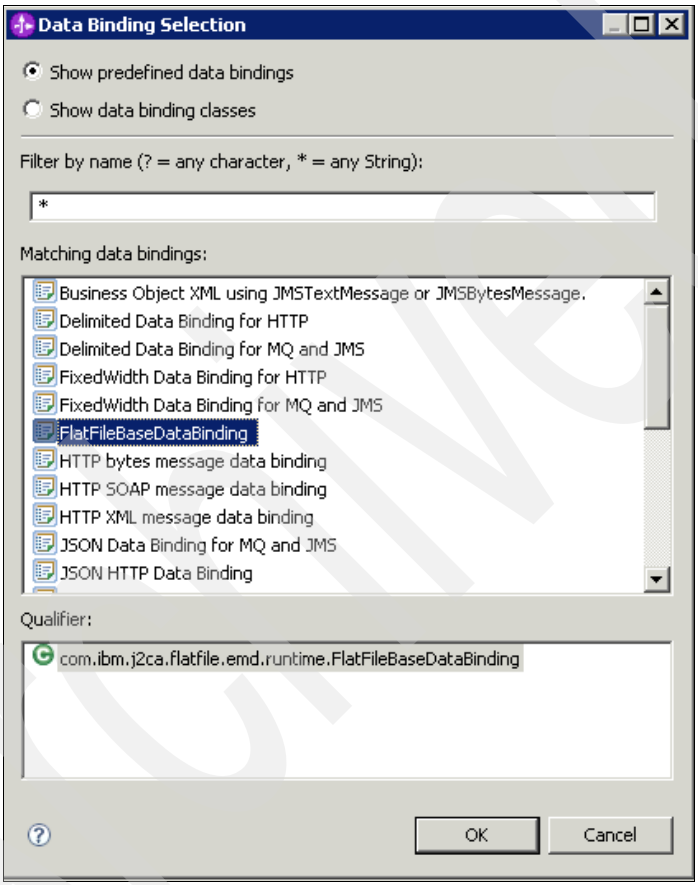

Figure 8-64 Data Binding Selection window

d. Back in the Binding Resource Configuration - Select a Configuration Type window (Figure 8-65), verify that Data binding class name is set to `com.ibm.j2ca.flatfile.emd.runtime.FlatFileBaseDataBinding` and click **Next**.

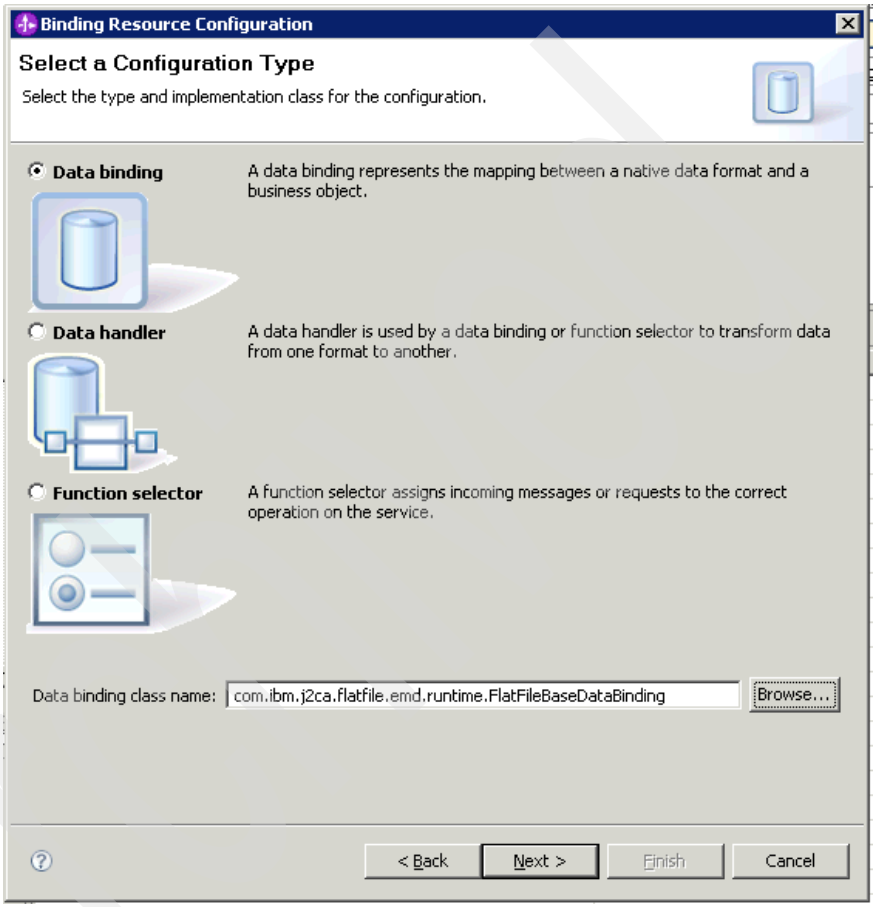

Figure 8-65 Binding Resource Configuration - Select a Configuration Type window

4. In the Binding Resource Configuration - Data Handler Properties window:
   a. For Binding type, select **DataHandler**.
   b. For Configured data handler, click **Browse**.
   c. In the Data Handler Selection window (Figure 8-66), select **WTXDataHandlerConfiguration**, which we created in the previous section, and click **OK**.

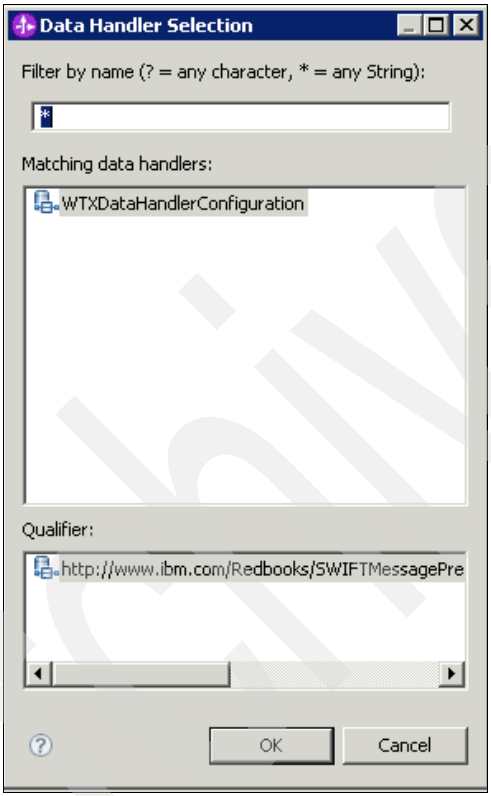

Figure 8-66 Data Handler Selection window

d. In the Binding Resource Configuration - Data Handler Properties window (Figure 8-67), verify the settings that you selected and click **Finish**.

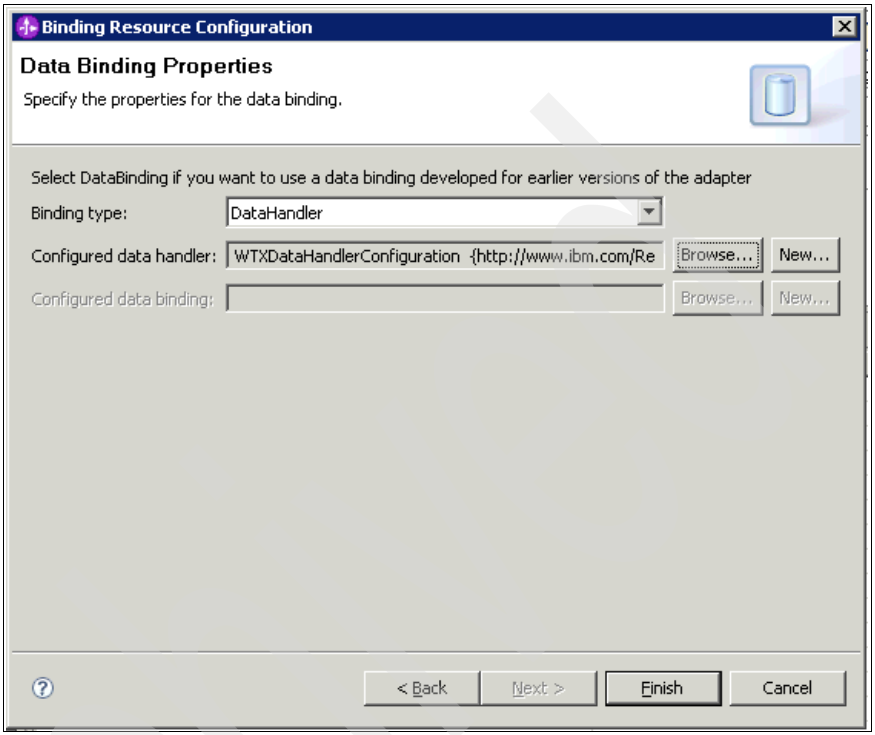

Figure 8-67 Binding Resource Configuration

5. Close the WTX data binding configuration.

Creating the directory structure for the scenario

Before we start to create inbound and outbound adapters, we must create the directory structure that is used in our scenario.

Directory structure in additional materials: Instead of creating the directory structure manually as explained in the following steps, you can extract the `ESB_Scenario_Directories.zip` file that is available through the additional materials that accompany this book. For more information about how to obtain the compressed file, see Appendix E, “Additional material” on page 749.

1. On the file system, create the `C:\$Redbook\ESB_Scenario` directory.
2. In the `C:\$Redbook\ESB_Scenario` directory, create the following folders:
   - MT103Input

     This folder is polled for new incoming MT103 messages.
   - SyntaxReject

     This folder is used to save all incorrect messages.
   - BICReject

     This folder is used to save all messages with incorrect BICs.
   - SWIFT_Gateway

     All the correct messages are saved in this folder in order to send them for further processing.

Creating the inbound and outbound adapters

Next we add and configure the inbound adapter. We perform similar steps to create three outbound adapters. In each case, we use the WebSphere Adapter for Flat Files.

The inbound adapter reads the file from the `C:\$Redbook\ESB_Scenario\MT103Input` directory and uses the WTXDataBindingConfiguration data binding configuration, created in the previous steps, to transform the MT103 message from its native SWIFT format to an MT103Msg business object.

The three outbound adapters save the message in their appropriate directories. The message is converted back from the business object to the SWIFT standard format.

Creating the inbound adapter

To create the inbound adapter:

1. Open the **ESB_Scenario1** assembly diagram.
2. In the Palette (Figure 8-68), expand the **Inbound Adapters** folder. Drag the **Flat File Adapter** icon to the white space in the assembly diagram.

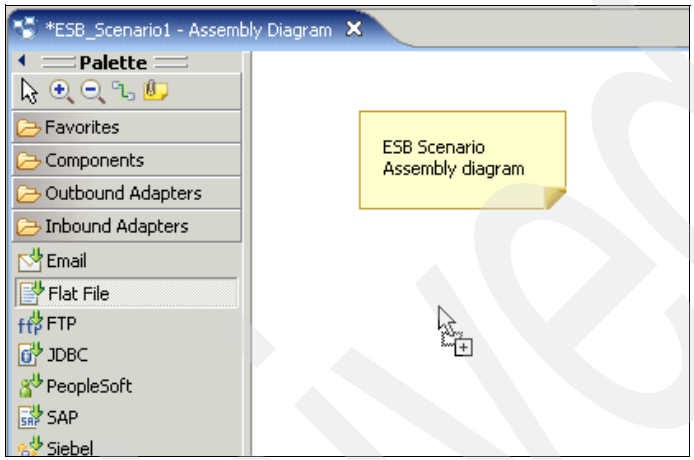

Figure 8-68 Adding a new Flat File inbound adapter instance

3. In the New Flat File Service wizard (Figure 8-69), select **Create a service (advanced)** and click **Next**.

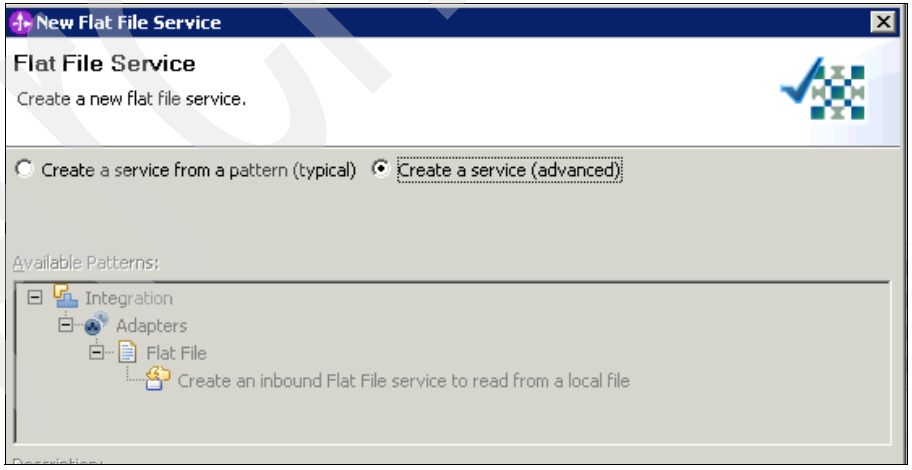

Figure 8-69 New Flat File Service wizard

4. In the External Service window (Figure 8-70), from the list of adapters, select **CWYFF_FlatFile** and click **Next**.

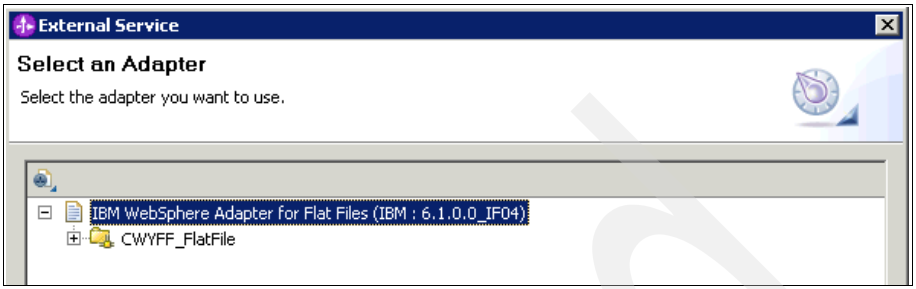

Figure 8-70 Selecting an adapter

5. In the next window:
   a. For Connection properties, in the Event directory field, click **Browse**.
   b. In the Browse For Folder dialog box (Figure 8-71), select **C:\$Redbook\ESB_Scenario\MT103Input** and click **OK**.

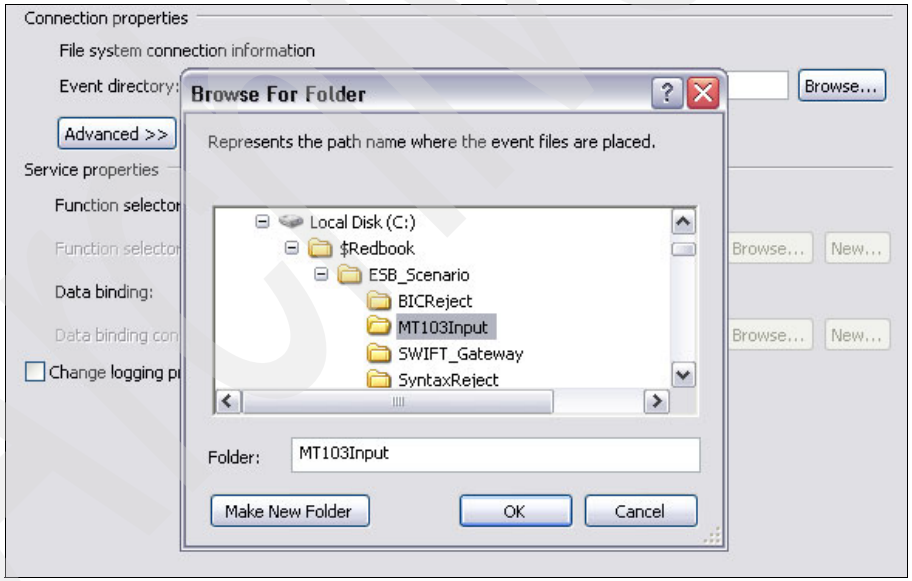

Figure 8-71 Browsing for the Event Directory

c. For Service properties (Figure 8-72):

   i. In the Data Binding field, select **Use a data binding configuration for all operations**.

   ii. In the Data binding configuration field, click **Browse**.

   iii. Select **WTXDataBindingConfiguration**.

d. Click **Next**.

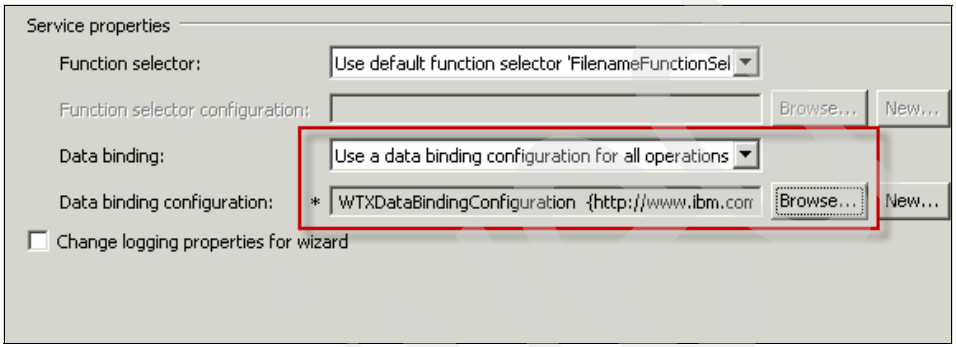

Figure 8-72 Setting the data binding properties for a new service

6. In the Operation window, click **Add**.

7. In the Add Operation window (Figure 8-73), for Data type for the operation, select **User defined type** and click **Next**.

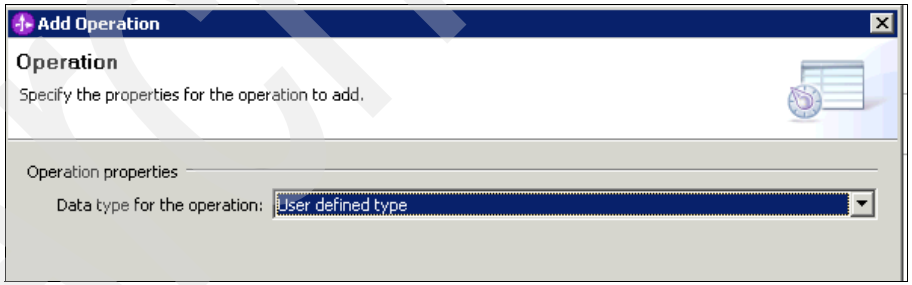

Figure 8-73 Specifying the operation properties

8. In the next Add Operation window:
   a. For Operation name, enter `CheckMT103Msg`. For Input type, click **Browse**.
   b. In the Data Type Selection window (Figure 8-74), select **MT103Msg**. Under Qualifier, make sure that you have chosen the correct schema file, which is **Definitions\MT103Msg.xsd** in this example. Click **OK**.

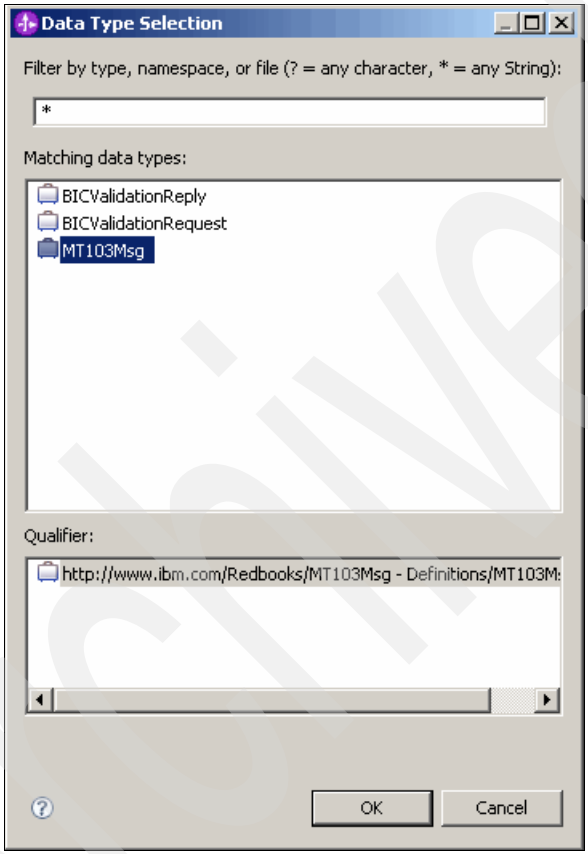

Figure 8-74 Choosing an input type for the new operation

c. Back in the Add Operation window (Figure 8-75), for Data binding, select **Use data binding configuration 'WTXDataBindingConfiguration'** and click **Finish**.

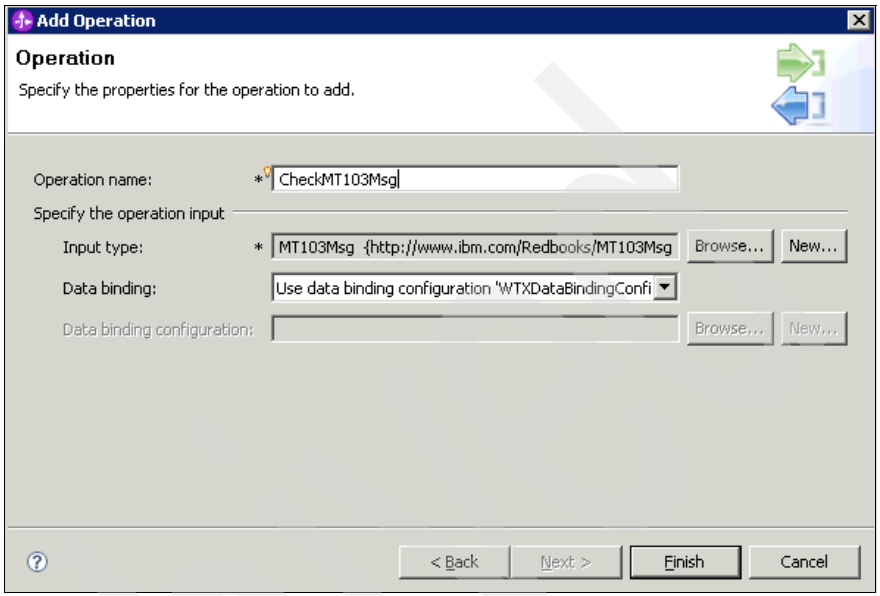

Figure 8-75 Specifying properties for the new operation

9. In the External Service - Operations window (Figure 8-76), verify your operation and click **Next**.

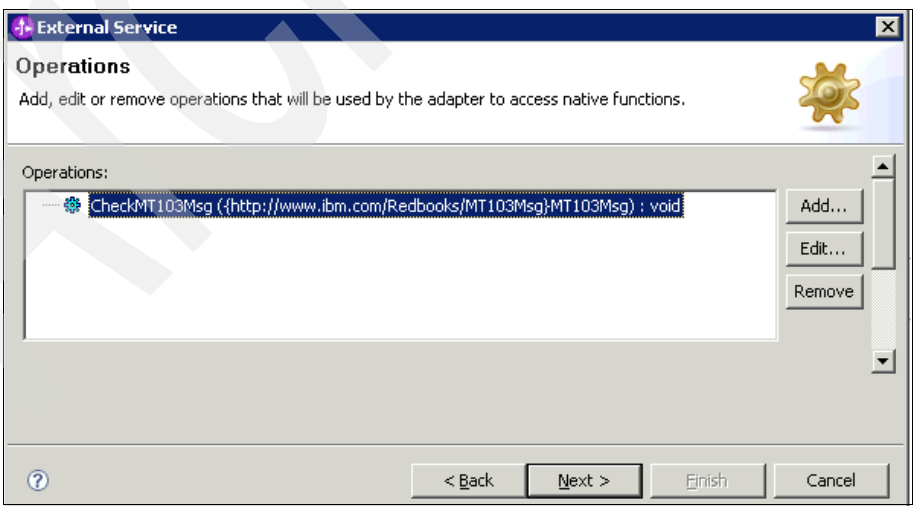

Figure 8-76 External Service - Operations window

10. In the External Service - Generate Service window (Figure 8-77), for Namespace, enter `http://www.ibm.com/Redbooks/MT103Preprocessing` and for Name, type `MT103Preprocessing`. Then click **Finish**.

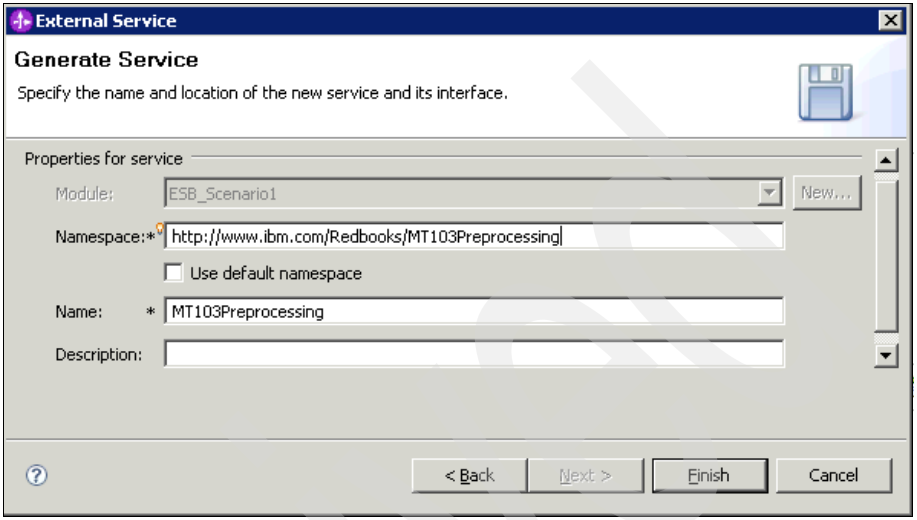

Figure 8-77 Finalizing the FlatFile service

The Flat File inbound adapter is now ready.

Creating the outbound adapters

In creating the outbound adapters, almost all steps are the same as for creating inbound adapters. We create an outbound service for saving valid MT103 messages in the `C:\$Redbook\ESB_Scenario\SWIFT_Gateway` directory.

All the files have the `MT103_Msg.#n.inp` name, where *#n* is a sequence number. The Flat File adapter uses the `C:\$Redbook\ESB_Scenario\sequence1.txt` file to store information about the actual file number.

To create the outbound adapters:

1. In the assembly diagram, expand the **Outbound Adapters** folder and drag a **Flat File adapter** to the canvas.
2. In the New Flat File Service wizard (Figure 8-69), select **Create a service (advanced)** and click **Next**.

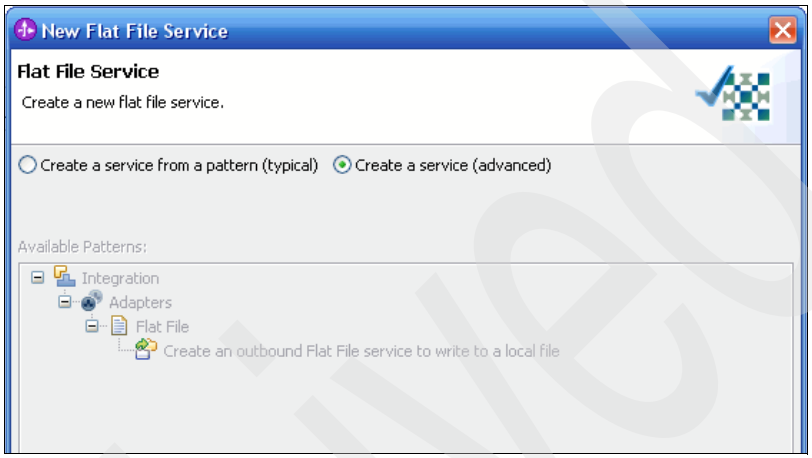

Figure 8-78 New Flat File Service wizard

3. In the External Service - Select an Adapter window (Figure 8-79), from the list of adapters select **CWYFF_FlatFile** and click **Next**.

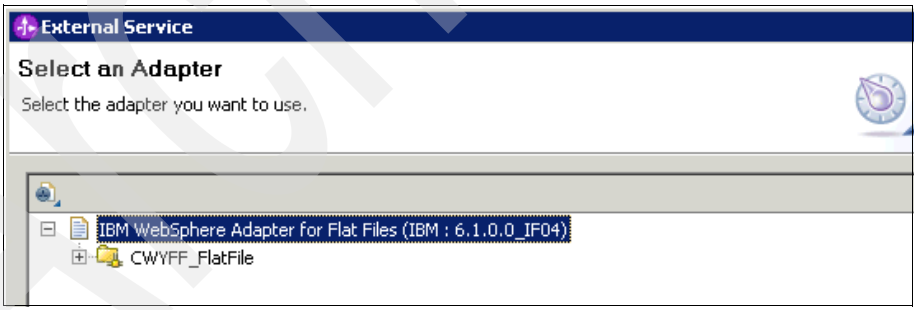

Figure 8-79 Selecting the Flat File adapter

4. In the External Service - Service Configuration Properties window (Figure 8-80):
   a. For Output directory, type `C:\$Redbook\ESB_Scenario\SWIFT_Gateway`.
   b. Click the **Advanced** button and expand the **Advanced properties**:
      i. For Default target file name, type `MT103_Msg.inp`.
      ii. For Sequence file, specify `C:\$Redbook\ESB_Scenario\sequence1.txt`.

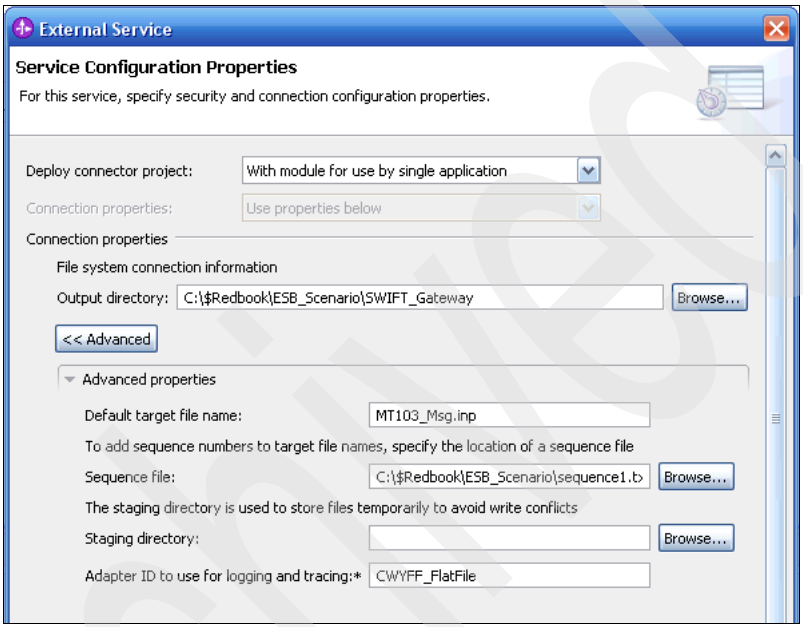

Figure 8-80 External Service - Service Configuration Properties window

c. Under Service Properties (Figure 8-81):

i. For Data binding, select **Use a data binding configuration for all operations**.

ii. For Data binding configuration, click **Browse** and select **WTXDataBindingConfiguration**.

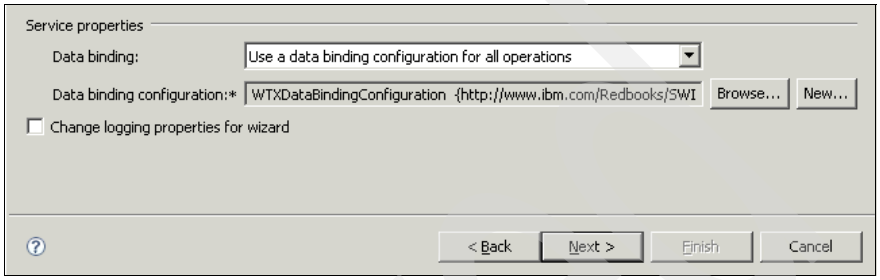

Figure 8-81 Service properties

d. Click **Next**.

5. In the Operation window, click **Add**.

6. In the Add Operation window (Figure 8-82), for Operation kind, select **Create**. For Data type for the operation, select **User defined type**. Then click **Next**.

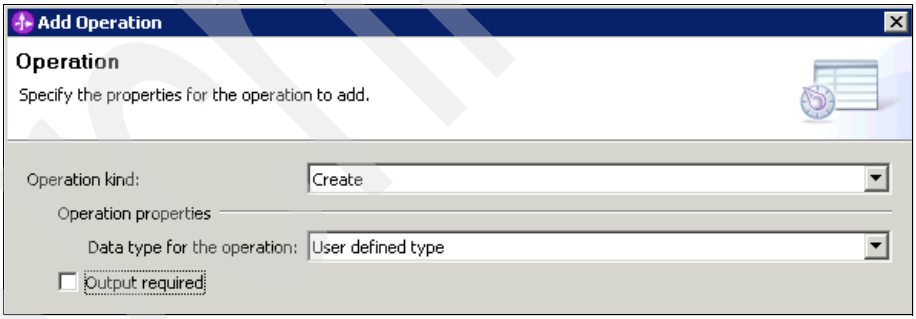

Figure 8-82 Adding a new operation to the outbound service

7. Leave Operation as `create`, and browse for business object **MT103Msg** as input for the operation (see step 8 on page 546). Use the data binding configuration **WTXDataBindingConfiguration**. Click **Finish**.

8. In the External Service - Operations window (Figure 8-83):
   a. For the Output directory, enter `C:\$Redbook\ESB_Scenario\SWIFT_Gateway`.
   b. Under Advanced properties, for Default target file name, type `MT103_Msg.inp`.
   c. Ensure that the **Generate a unique file** option is cleared.
   d. Click **Next**.

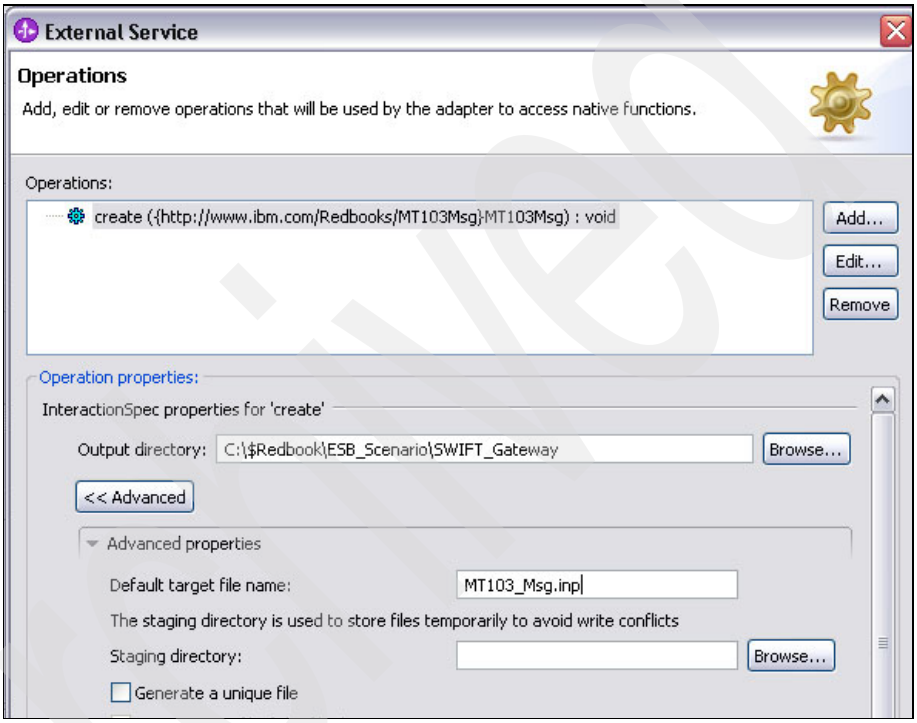

Figure 8-83 Completing the outbound service

9. In the External Service - Generate Service window (Figure 8-84), for Namespace, type `http://www.ibm.com/Redbooks/ProcessValidMessage`, and for Name, type `ProcessValidMessage`. Click **Finish**.

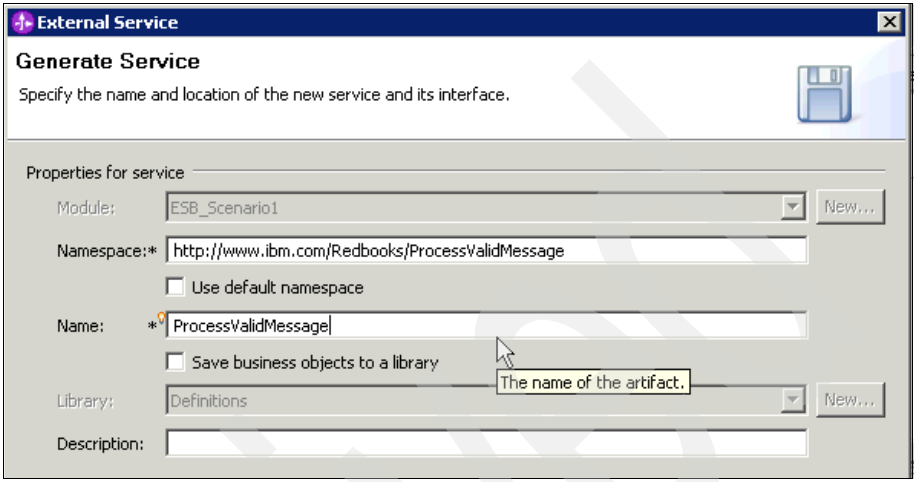

Figure 8-84 Completing the outbound service

10. Repeat each of these steps for the RejectBIC and Reject Syntax outbound services. Change the appropriate properties as shown in Table 8-3.

Table 8-3 Specification of the outbound service properties

Service name	ProcessValidMessage	RejectBIC	RejectSyntax
Output folder	C:\$Redbook\ESB_ Scenario\SWIFT_Gateway	C:\$Redbook\ESB_ Scenario\BICReject	C:\$Redbook\ESB_ Scenario\SyntaxReject
Default file name	MT103_Msg.inp	MT103_BIC.err	MT103_Syntax.err
Sequence file	C:\$Redbook\ESB_ Scenario\sequence1.txt	C:\$Redbook\ESB_ Scenario\sequence2.txt	C:\$Redbook\ESB_ Scenario\sequence3.txt
Namespace	http://www.ibm.com/ Redbooks/Process ValidMessage	http://www.ibm.com/ Redbooks/RejectBIC	http://www.ibm.com/ Redbooks/RejectSyntax

Figure 8-85 shows the completed assembly diagram after finishing the steps in this task.

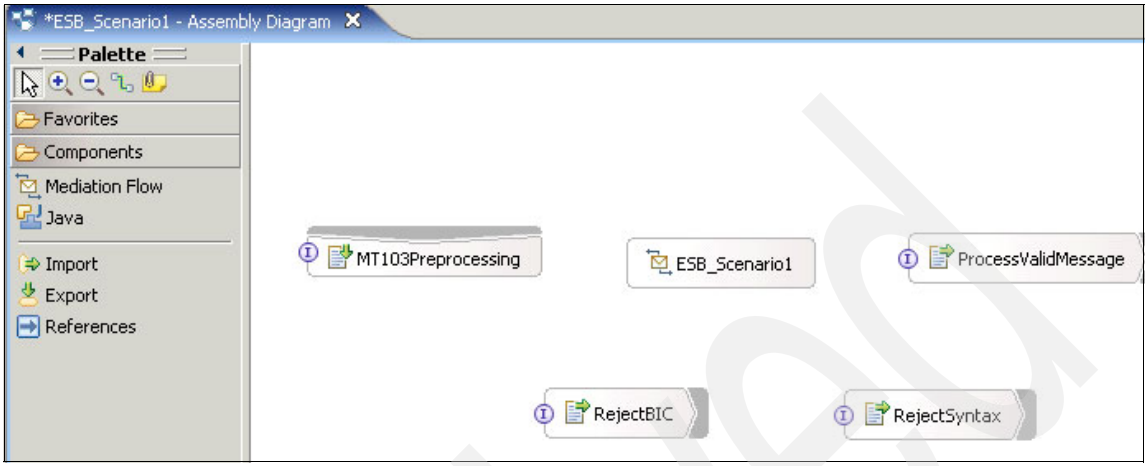

Figure 8-85 Assembly diagram after adding the Flat File adapters

Completing the assembly diagram

Before starting to implement the mediation flow, we must complete the assembly diagram. In our scenario description, we do not describe in detail the implementation of BICValidationService. We prepare a simple implementation of the mentioned service that you can download from the sample materials. For details, see Appendix E, “Additional material” on page 749.

To complete the assembly diagram:

1. Import the BICValidationService implementation from the additional materials as explained in Appendix E, “Additional material” on page 749.
   a. Select **File** → **Import**.
   b. In the Import wizard, select **Project Interchange**.

c. In the Import Project Interchange Contents window (Figure 8-86):

   i. For From zip file, click **Browse** and navigate to the **ESB_Scenario_Services.zip** file that you downloaded.

   ii. From the list of projects select **Ports, Services, ServicesApp, ServicesEJB,** and **ServicesWeb**.

   iii. Click **Finish**.

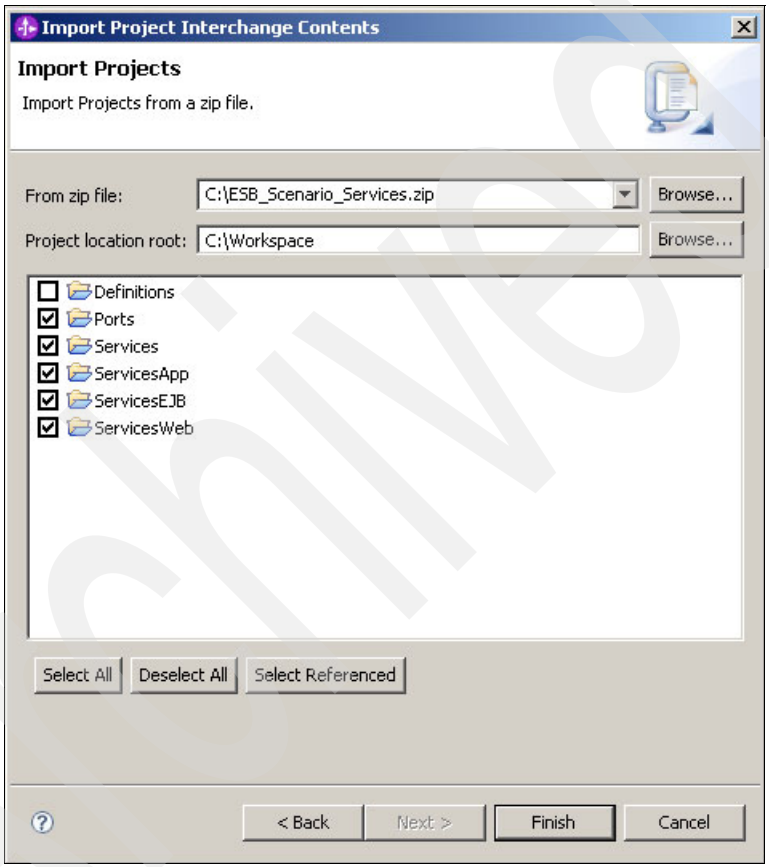

Figure 8-86 Importing the service implementation

2. Right-click the **ESB_Scenario1** mediation module and select **Open Dependencies**. On the Dependencies page (Figure 8-87), add the imported library **Ports** to Libraries section. Then save and close the editor.

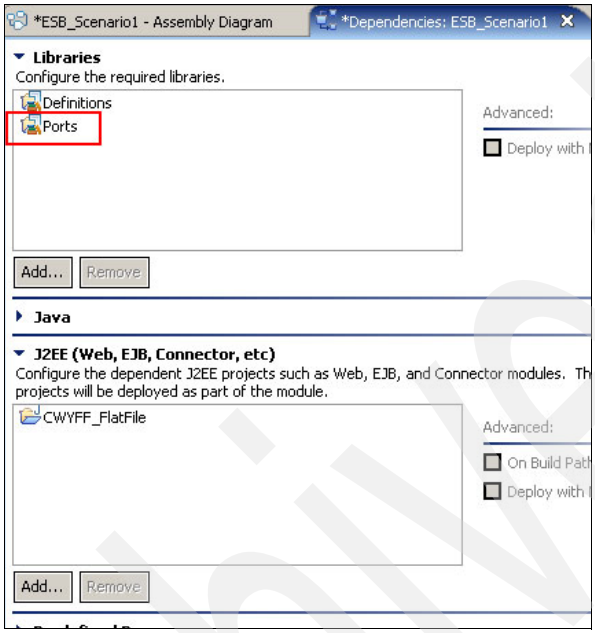

Figure 8-87 Adding the library to the dependencies

3. Navigate to the **Ports** library and under Web Service Port, double-click **BICValidationServiceExport1_BICValidationServiceHttpService** (Figure 8-88).

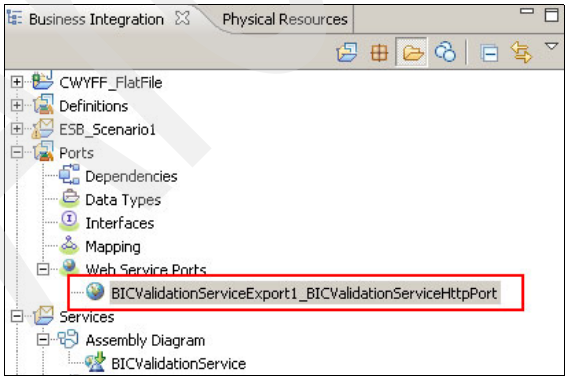

Figure 8-88 Web service port

4. In the interface editor (Figure 8-89), click **BICValidationServiceExport1_ BICValidationServiceHttpPort**. Verify that the address of the service (port number) is appropriate for your ESB server settings. You can check the port number that is used by your WebSphere ESB instance in the console or in the server logs. If necessary, change the server name or port number.

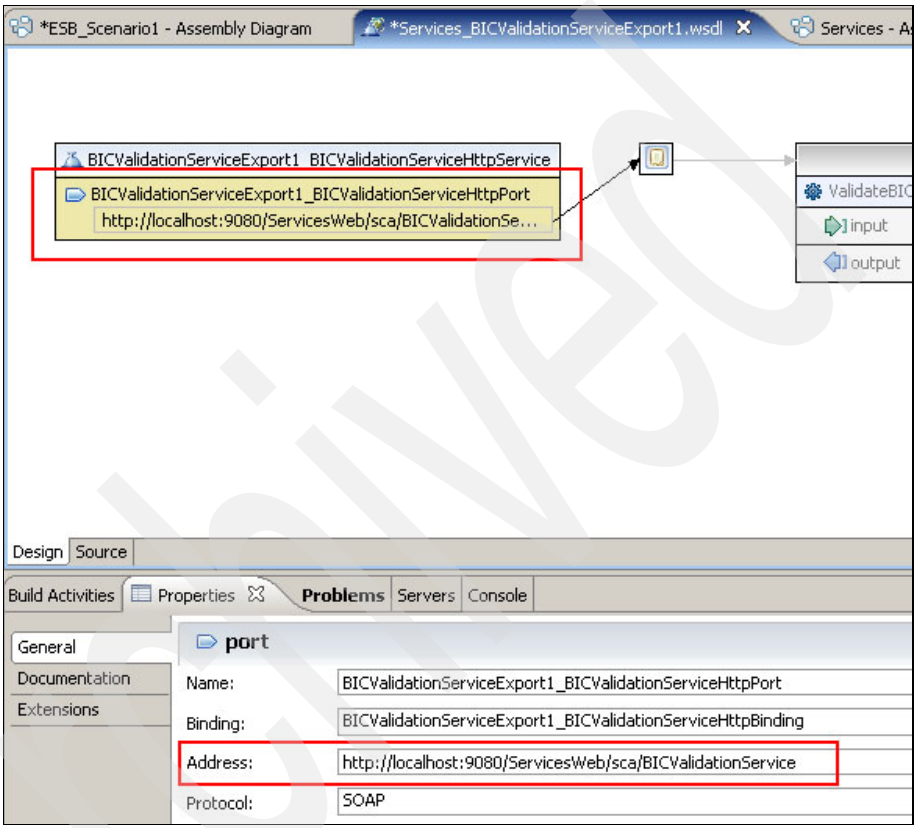

Figure 8-89 Settings for the address of the Web service

5. Add the BICValidationService import to the ESB_Scenario1 assembly diagram:
   a. Open the assembly diagram for the **ESB_Scenario1** project.
   b. In the Business Integration Navigator, expand the **Web Service Ports** folder in the just imported **Ports** library.
   c. Click and drag the **BICValidationServiceExport1_ BICValidationServiceHttpPort** port to the **ESB_Scenario1** diagram.

d. In the Component Creation dialog box (Figure 8-90), select **Import with Web Service Binding**. Click **OK**.

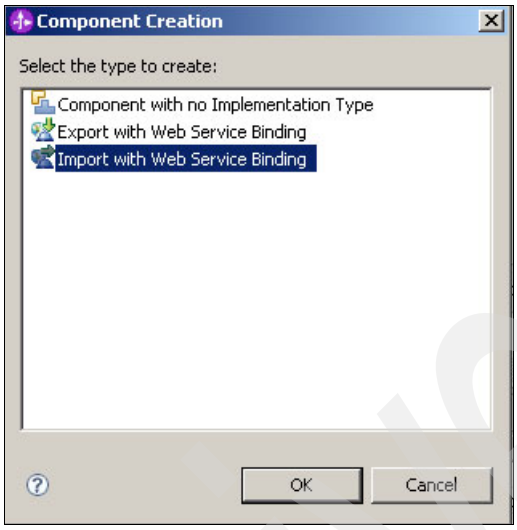

Figure 8-90 Importing the Web Service

A new SCA component called *BICValidationServiceImport1* is created as shown in Figure 8-91.

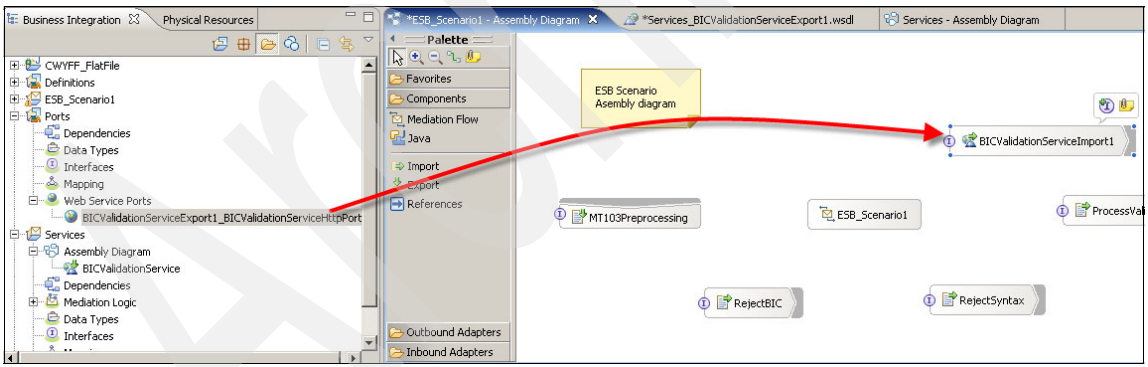

Figure 8-91 New SCA component

e. Save the assembly diagram.

6. Add the wires between the components:
   a. Connect the **MT103Preprocessing** export with the **ESB_Scenario1** module (Figure 8-92).

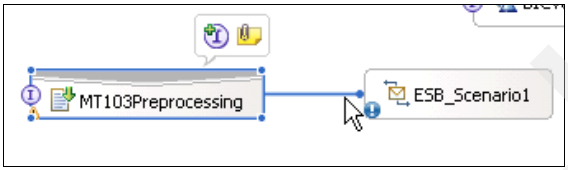

Figure 8-92 Connecting the components

   b. When prompted about adding a matched interface, click **OK**.
   c. Connect the **ESB_Scenario1** module with the following import components:
      - BICValidationServiceImport1
      - ProcessValidMessage
      - RejectSyntax
      - RejectBIC
   d. For each time that you are prompted to add a matching reference, click **OK**.
   e. Save the assembly diagram.

Figure 8-93 shows how the assembly diagram looks after completing this task.

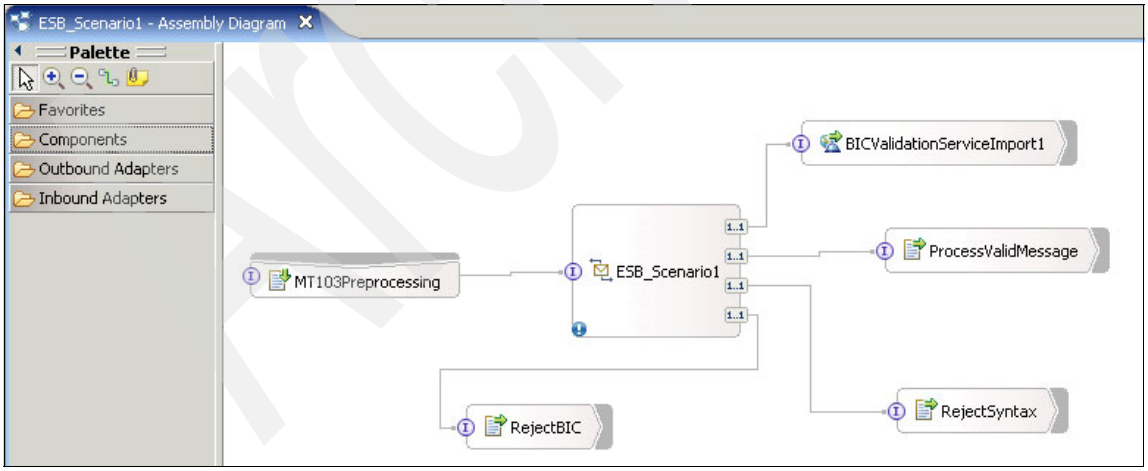

Figure 8-93 Assembly diagram after connecting all the players

Completing the implementation of the mediation flow

To implement the ESB_Scenario1 mediation flow:

1. When a message in the form of a business object comes to the input, choose either of the following actions:
   - If the Status field in the business object is equal to `SYNTAX ERROR` (which is set by the WebSphere Transformation Extender map), save the message in the SyntaxReject folder, by calling the create method from the RejectSyntax service.
   - If the Status field in the business object does not indicate a syntax error, prepare a request for the BIC Validation Service.
2. Call the BIC Validation Service. Choose one of the following options:
   - If the reply from the service has a return code of 0, the BIC is valid. In this case, set the Status field to `BIC OK`.
   - If the reply from the service has a return code other than 0, set the Status field to `BIC ERROR`.
3. Depending on the option that you selected in step 2, choose the appropriate following option:
   - If the Status field is equal to `BIC OK`, send the message for further processing by calling a create method from the ProcessValidMessage service.
   - If the Status field is *not* equal to `BIC OK`, save the message in the BICReject folder by call the create method from the RejectBIC service.

To prepare the implementation of the ESB_Scenario1 mediation flow:

1. Right-click the **ESB_Scenario1** component and select **Generate Implementation** (Figure 8-94).

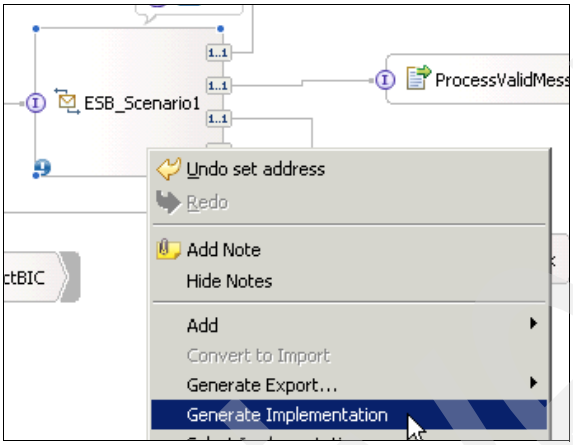

Figure 8-94 Generate Implementation of the Mediation Flow

2. Click **OK** to confirm the main folder as the place to store the implementation.
3. In the Mediation Flow Editor that opens with the MT103Preprocessing interface, implement the CheckMT103Msg operation. You should see that four references have been created.
4. In the Operation connections section, connect the **CheckMT103Msg** operation to the MT103Preprocessing interface with the following operations:
   - create from **ProcessValidMessagePartner**
   - create from **RejectSyntaxPartner**
   - create from **RejectBICPartner**

Figure 8-95 shows how the Mediation Flow Editor should look after completing this step.

Tip: If you do not see the mediation flow, click the connectors between CheckMT103Msg and the references.

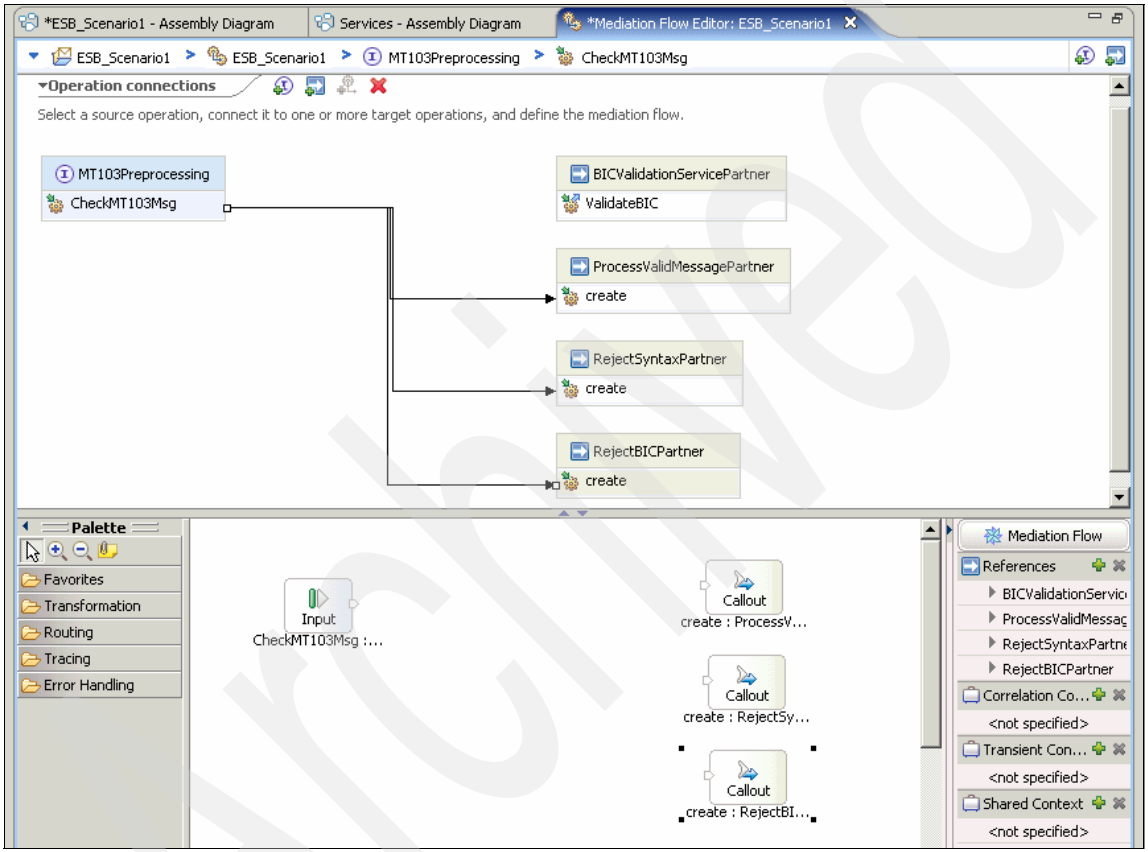

Figure 8-95 The ESB_Scenario1 mediation flow before the implementation

5. Add the input node. In the first step of our flow, we saved the Object input to the log. The messages that go through the flow are displayed in the Console view:

   a. Navigate to the **Palette**.

   b. From the Transformation folder in the Palette, drag the **Custom Mediation** primitive to the canvas.

c. In Description section, on the **Properties** tab, change Display name to `WriteMessageToLog`. The Name field should change automatically.

d. Connect the **Input** node with **WriteMessageToLog** node's input terminal (Figure 8-96).

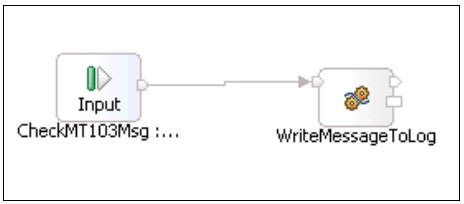

Figure 8-96 Connections of nodes

e. On the **Properties** tab of the WriteMessageToLog node:

i. In the Terminal section, under Output terminal, click **out** (Figure 8-97).

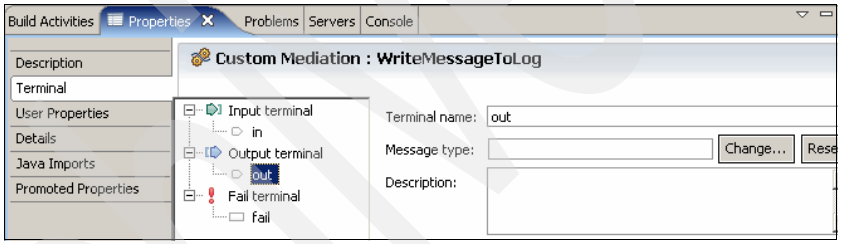

Figure 8-97 Changing the out terminal

ii. For Message type, click the **Change** button.

iii. In the Change Message Type window (Figure 8-98), for Interface, click **Browse** and select the **MT103Preprocessing** interface. Click the list options to select the Operation, Message Category, and Message Type. Click **OK**.

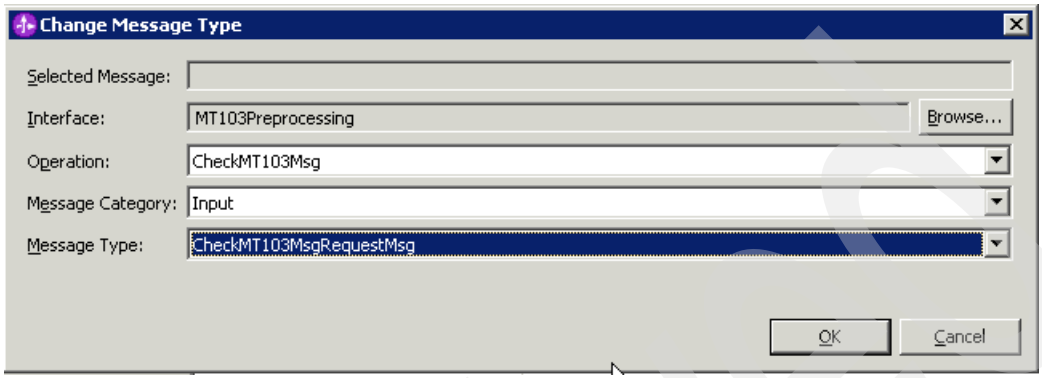

Figure 8-98 Change Message Type window for the terminal

f. In the Details section, switch the implementation from Java to **Visual**.

g. If you are prompted about changing the type of snippet, click **Yes**.

h. From the palette on the left, select **Standard Visual Snippet**.

i. From the Add a Standard Visual Snippet dialog box, open the utility folder and select the **print BO to log** snippet. Click **OK**.

j. Drag the snippet to the canvas *before* the existing fire node.

k. Connect the **smo** input object with the **print BO to log** node.

l. Verify your custom mediation node (Figure 8-99) and save it (press Ctrl+S).

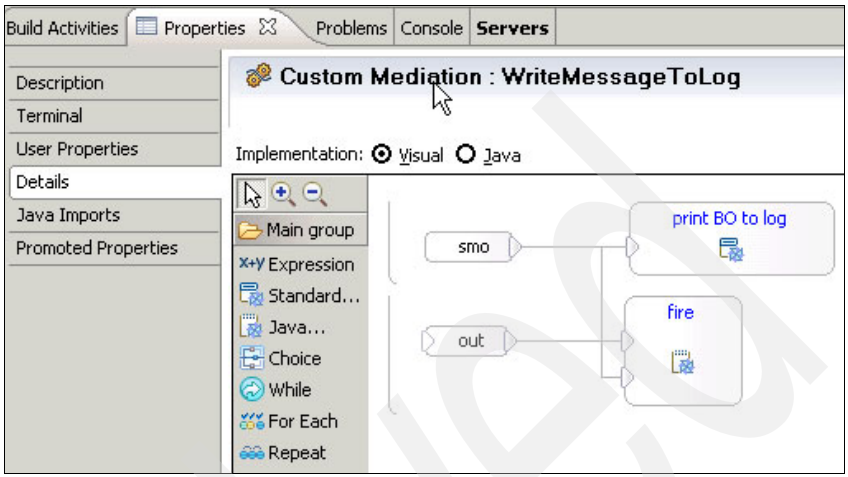

Figure 8-99 Custom Mediation primitive - Visual editor

6. Back on the Mediation Flow Editor, create a new filter node to ensure that the incoming message has no syntax errors:
   a. From the Palette, select a Message Filter Node (from the Routing drawer) and drag it to the diagram. Name the node `CheckIfNoSyntaxError`.
   b. Connect the WriteMessageToLog node's **out** terminal with the **in** terminal of the new node.
   c. In the Properties of the CheckIfNoSyntaxError node:
      i. Under the Terminal section, rename the Output terminal match1 to `SyntaxError`.
      ii. Click the **Details** tab and click the **Add** button to add a new filter.
   d. In the Add/Edit Properties window, click the **Edit** button to edit the Pattern for your Filter.
      i. Use the XPath Expression® Builder to build a condition to check the Status field in the MT103Msg object. We route all the messages with syntax errors to the SyntaxError terminal. Therefore, the condition should check whether the value of Status field is equal to SYNTAX ERROR.

Figure 8-100 shows the created expression.

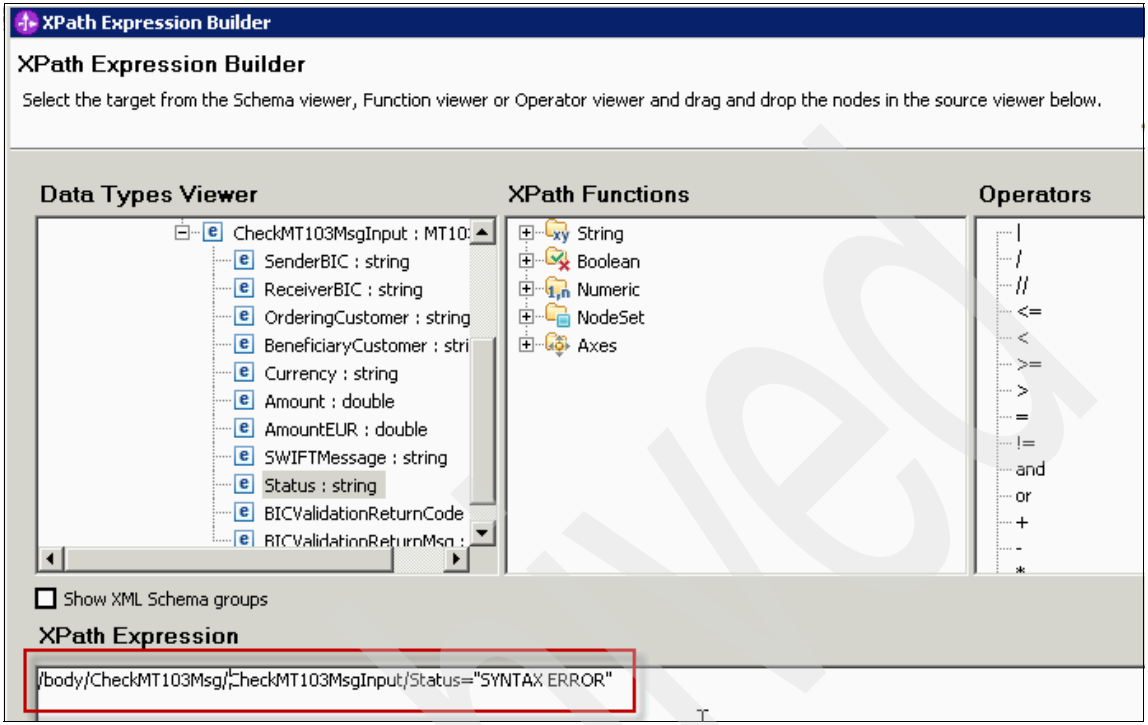

Figure 8-100 Building the XPath Expression

ii. In the XPath Expression Builder window, click **Finish**.

iii. In the **Add/Edit Properties window**, click **Finish**.

e. Press Ctrl+S to save.

7. Prepare the Callout of the **RejectSyntax** service.

In all of input/output operations, we use the same type for input parameter, which is the MT103Msg business object. However each operation has its own interface. Therefore, we must use a business object map to translate the message from one context to another.

a. Back on the Mediation Flow Editor, in the Palette, open the Transformation folder and add the **Business Object Map** primitive to your flow.

b. Change the name of the new node to `PrepareOutput`.

c. Wire the SyntaxError terminal of the **CheckIfNoSyntaxError** node with the input terminal of the **PrepareOutput** Node.

d. Connect the **out** terminal of the **PrepareOutput** Node with the **in** terminal of the **Callout** node for the **create:RejectSyntaxPartner** operation. Figure 8-101 shows the mediation flow at this point.

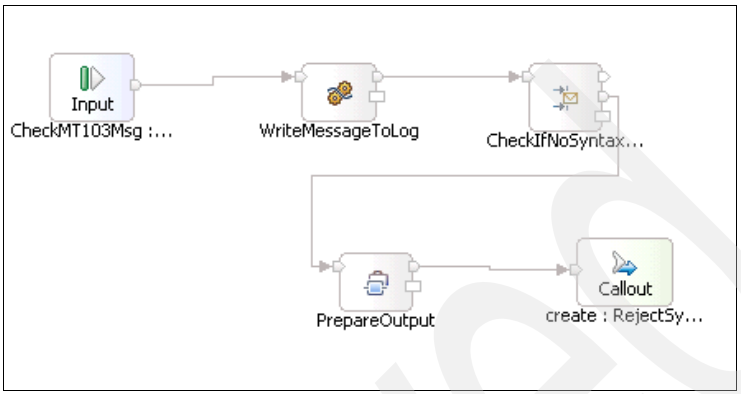

Figure 8-101 The ESB_Scenario1 mediation flow

e. Double-click the **PrepareOutput** node.

f. In the New BO Map - Business Object Map window (Figure 8-102), for Namespace, type `http://www.ibm.com/Redbooks/ESB_Scenario1`. For Name, type `ConvertToOutput` for your new map. Then click **Next**.

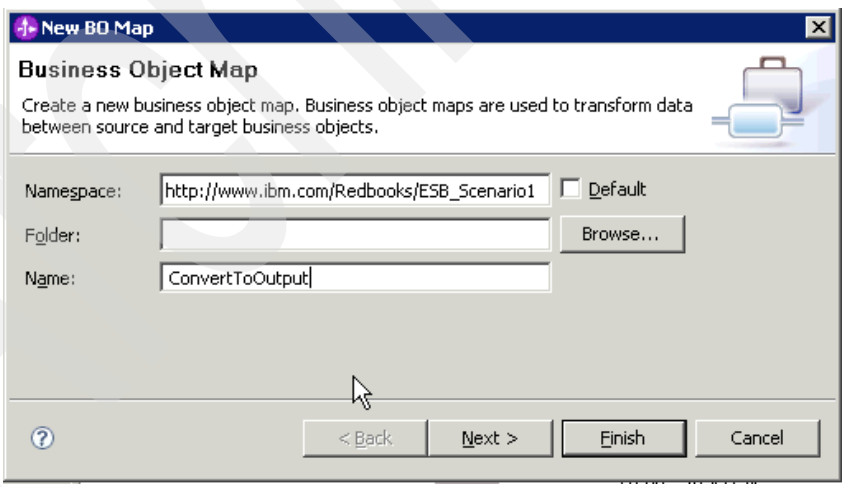

Figure 8-102 New BO Map - Business Object Map window

g. In the New BO Map - Specify Message Types window (Figure 8-103), confirm the default values and click **Finish**.

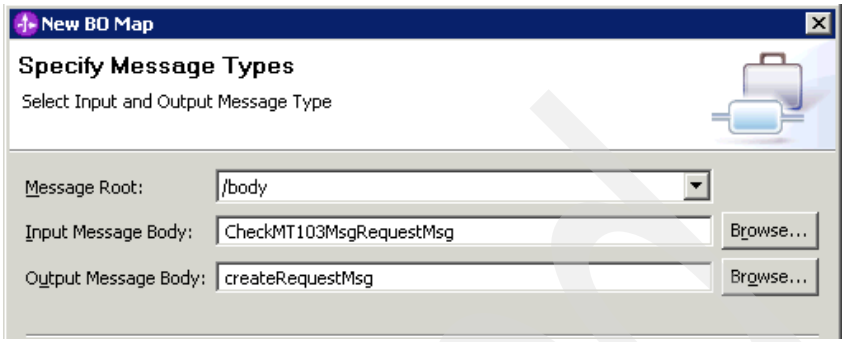

Figure 8-103 New BO Map - Specify Message Types window

h. In the Business object map editor, expand both the input and output objects and wire both the **MT103Msg** objects by dragging **CheckMT103MsgInput** to **createInput** (Figure 8-104).

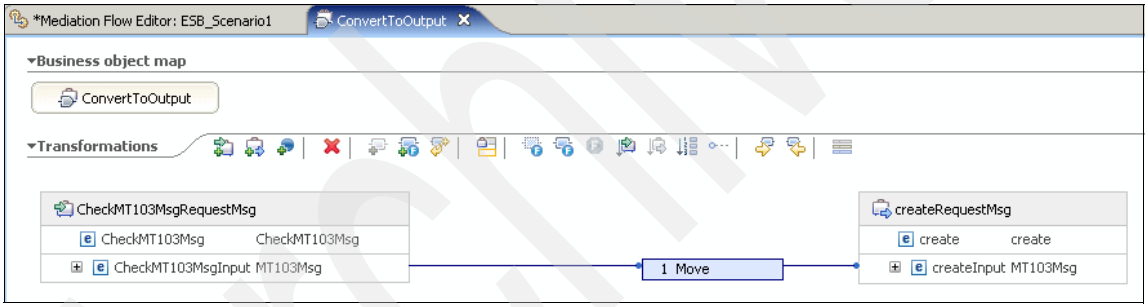

Figure 8-104 Business object map

i. **Save** and close the Business Object Map Editor.

j. **Save** the Mediation Flow Editor and the assembly diagram.

Now we must prepare the invocation of the BICValidation service. The input for the ValidateBIC operation is the BIC. See step 6 on page 494 where we created the interface for this service.

Again we use the business object map nodes. Before we start to create new business object maps, we must prepare a place to store the input MT103Msg object. If we do not do this, we might lose our input message, which we need later in the mediation flow during the service call. We use the transient context to store the data.

8. Add the MT103Msg element to the Transient Context of the mediation flow:
   a. In the Mediation Flow Editor, go to the tray on the right side of the editor (Figure 8-105), and click the plus (+) button next to Transient Context.

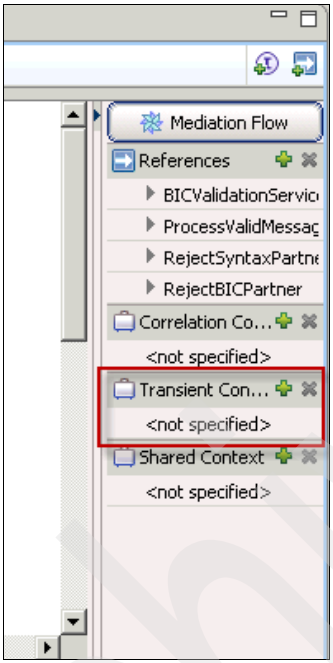

Figure 8-105 Specifying the Transient Context

   b. Select **MT103Msg** business object from the list. Click **OK**.
   c. Save the mediation flow (Ctrl+S).

9. Prepare the BICValidation service request:
   a. In the Mediation Flow Editor, from the Routing folder, select the **Service Invoke** node and drag it to the diagram.
   b. In the Select Reference Operation dialog box (Figure 8-106), select the **ValidateBIC** operation from the **BICValidationServicePartner** reference and click **OK**.

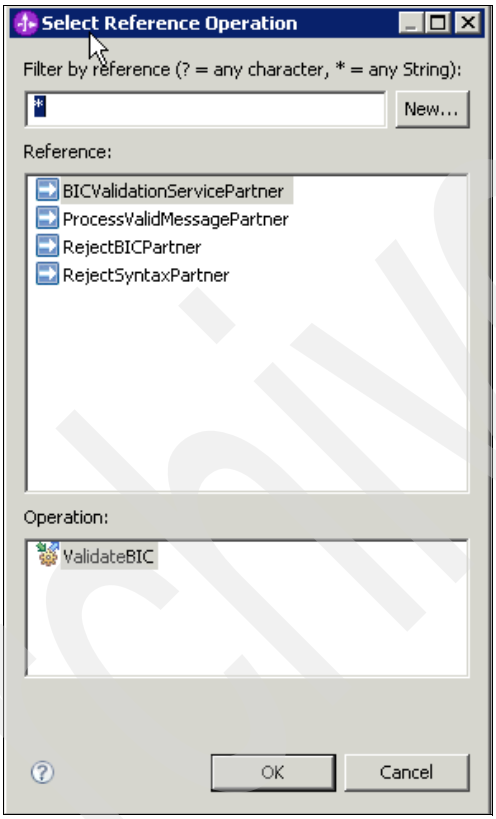

Figure 8-106 Select Reference Operation dialog box

   c. Rename the node to `CallBICValidationService`.
   d. Add a new Business Object Map node and name it `PrepareServiceRequest`.
   e. Connect the **default** terminal from the **CheckIfNoSyntaxError** node with the **in** terminal of the **PrepareServiceRequest** node.

f. Connect the **out** terminal of the **PrepareServiceRequest** with the **in** terminal of the **CallBICValidationService** node.

Figure 8-107 shows the flow after completing these steps.

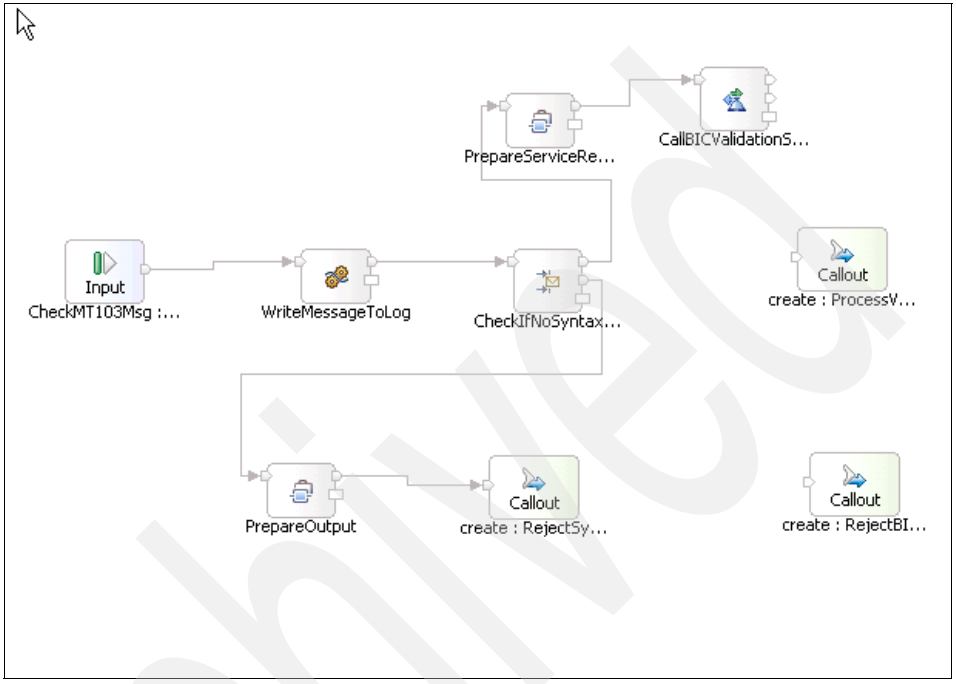

Figure 8-107 The ESB_Scenario1 Mediation Flow

g. Double-click the **PrepareServiceRequest** node to generate a new business object map.

h. In the New BO Map window (Figure 8-108), for Name, type `PrepareServiceRequest`, and for Namespace, type `http://www.ibm.com/Redbooks/ESB_Scenario1`. Then click **Next**.

Figure 8-108 New BO Map window

i. On the Specify Message Types page, change the Message Root to a forward slash (/) and click **Finish**.

> **Important:** You must change Message Root from /body to `/ in Root` to access the message context.

j. Complete the mapping in the Business Object Map Editor.

   i. Move the **MT103Msg** element from the /body of the Input Object to the **context/transient** element of the Output Object.

   ii. Move the **ReceiverBIC** element from the Transient context of the output (!) to the **/body/ValidateBIC/Request/BIC** element, also in the output (Figure 8-109).

> **Note:** In the second mapping rule, we copied from the TransientContext one **Output** to the BIC on the **Output.**

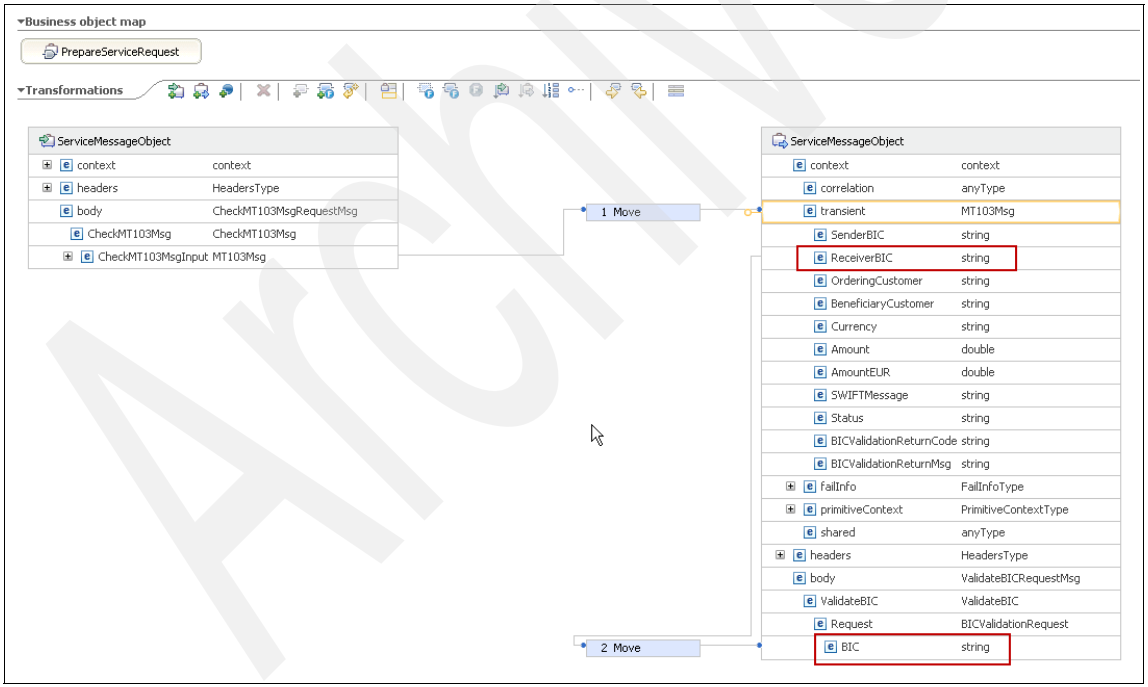

Figure 8-109 Business Object Map Editor

iii. Save and close the business object map.

k. In the Message Flow Editor, add a new Business Object Map node and name it `CopyServiceReply`. Connect the **out** terminal of the **CallBICValidationService** to the **in** terminal of the **CopyServiceReply** node.

l. In the Properties view of the CopyServiceReply node, on the **Terminal** tab, change the Message type for the **out** terminal to `{http://www.ibm.com/Redbooks/MT103Preprocessing}CheckMT103MsgRequestMsg` (Figure 8-110), which is the same way as in Figure 8-98 on page 564.

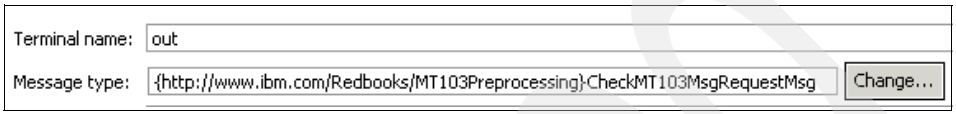

Figure 8-110 Message type for the out terminal

m. Double-click **CopyServiceReply**. In the New BO map wizard, create a map named `CopyServiceReply` in the namespace `http://www.ibm.com/Redbooks/ESB_Scenario1`. Again use the / value for Message Root.

n. In the Business Object Map Editor:

   i. Move the **MT103Msg** from **/context/transient** to **/body/CheckMT103Msg/CheckMT103MsgInput**.

   ii. Move the **ReturnCode** from **ValidateBICResponse** to **BICValidationReturnCode** in **MT103Msg**.

   iii. Move the **Description** from **ValidateBICResponse** to **BICValidationReturnMsg** in **MT103Msg**.

   iv. Create a custom mapping from the return code of the ValidateBICResponse to the Status field in MT103Msg. If the return code is equal to 0, set the value of the Status field to `BIC OK`. Otherwise, set the value to `BIC ERROR`.

First drag a line between the source (ReturnCode) and the target (Status) field. Then click the line again and select **Custom** (Figure 8-111).

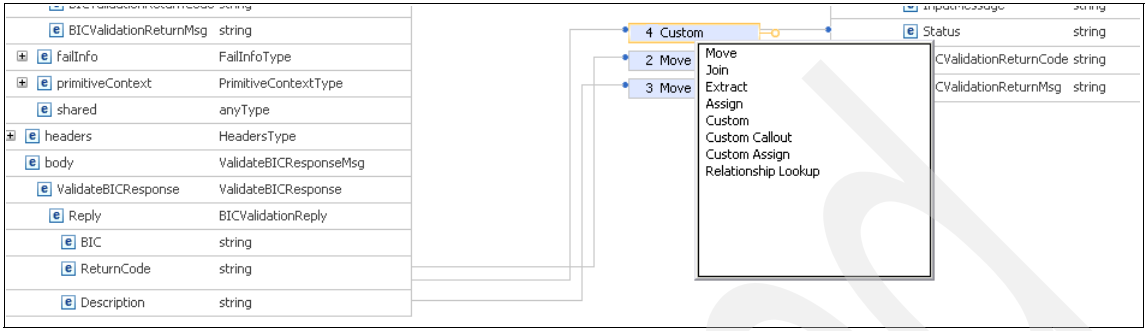

Figure 8-111 creating the custom mapping

v. On the **Details** tab, choose a Visual Implementation. First select an **x+y Expression** and drop it on the palette. Then click in it and select the message type (Figure 8-112).

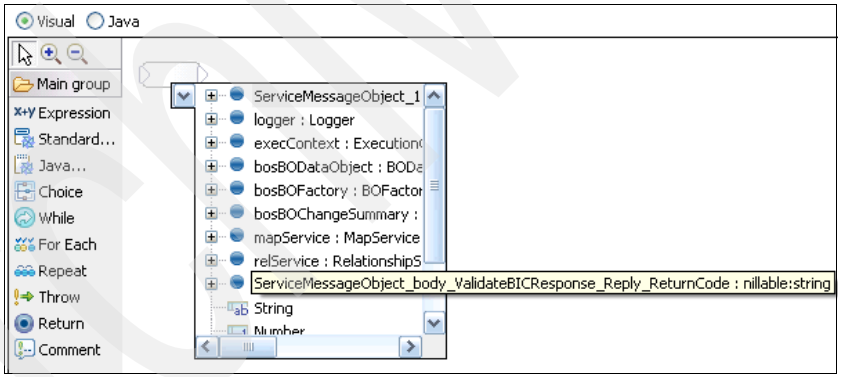

Figure 8-112 select the source object

Select another **x+y Expression**, drop it on the canvas, and choose **String**. In the window, type 0.

vi. Add a Standard Visual Snippet. The type is **text equal to**, which is available in the text folder.

vii. Connect both expression snippets to the **text equal to** snippet (Figure 8-113).

Figure 8-113 The first part of the visual Java snippet

viii. Add a **Choice** snippet. In the “if true” section, drop an **x+y Expression snippet** that you configured as **BIC OK** and the variable to contain the status (available under Variables at the right side of the window). In the “otherwise” section, drop another **x+y Expression snippet** that you configured as BIC ERROR and the same variable as for the previous snippet. Then wire them. Figure 8-114 shows how the right side of the expression should look.

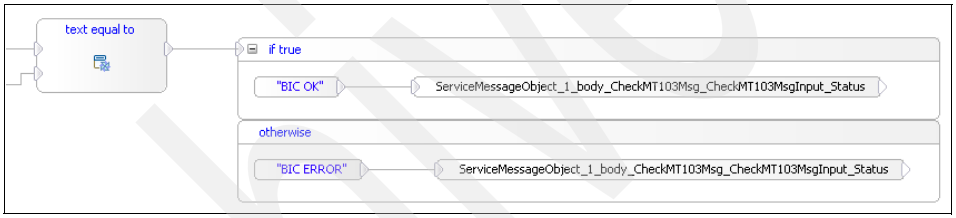

Figure 8-114 Right side of the visual Java snippet

Figure 8-115 shows the complete visual Java snippet.

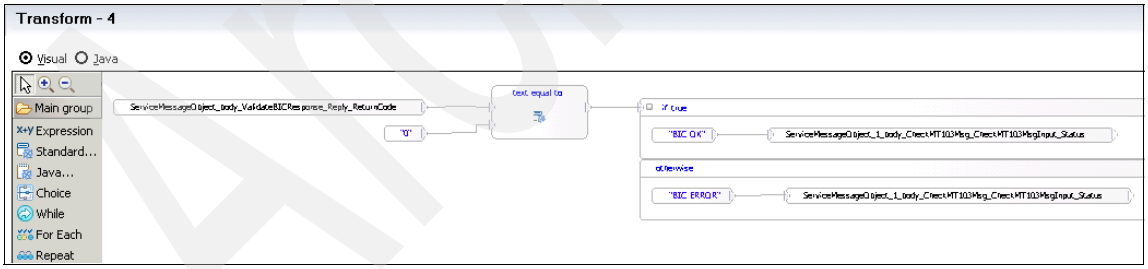

Figure 8-115 The complete visual Java snippet

ix. Verify that your CopyServiceReply map looks as shown in Figure 8-116 and save and close the map.

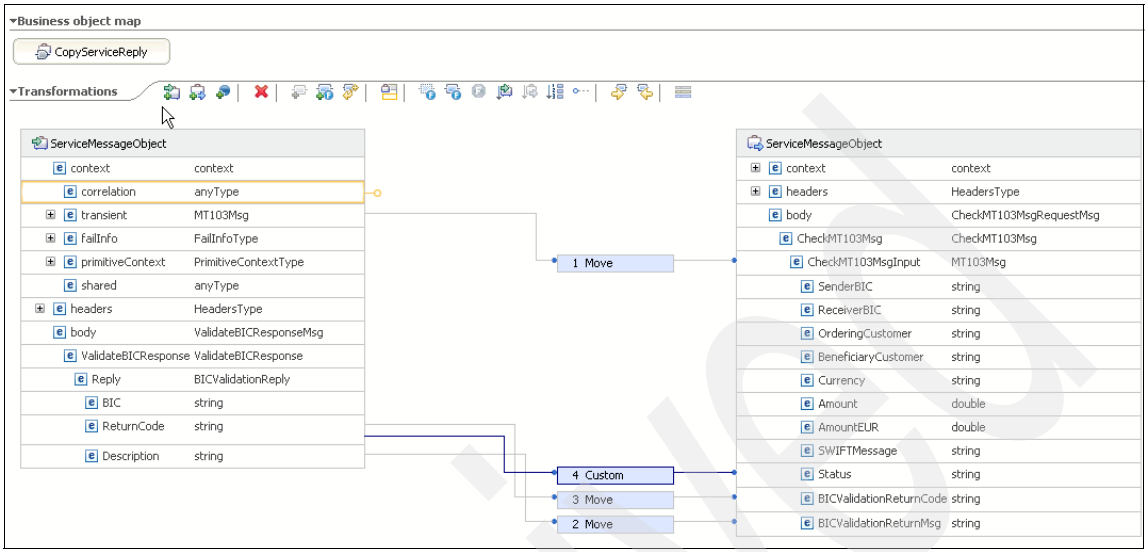

Figure 8-116 Business object map editor

Figure 8-117 shows how the mediation flow looks now.

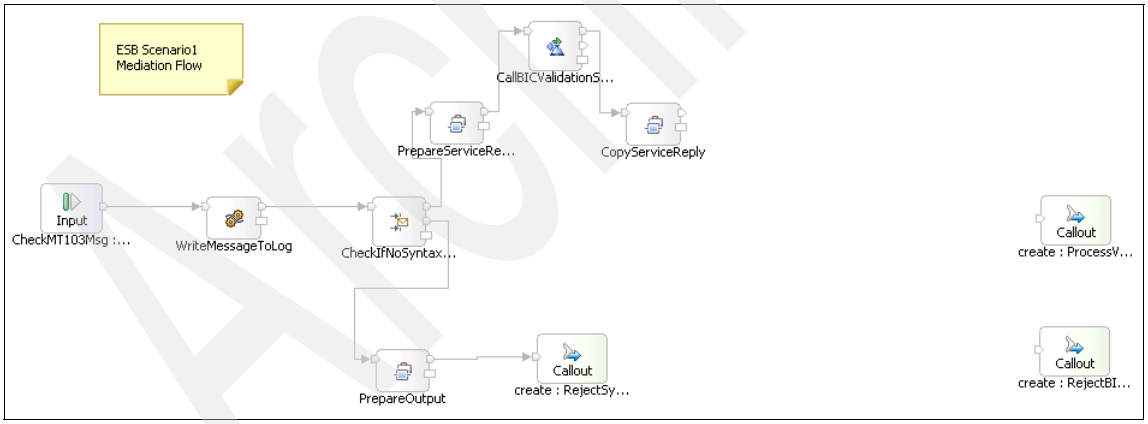

Figure 8-117 Mediation flow before adding the last nodes

10. Add three more nodes, which are all similar to the others that you previously created. To add the missing elements:

    a. Add the new Message Filter node, called **CheckIfNoBICError**. Connect the **out** terminal of the **CopyServiceReply** node to its **in** terminal.

Configure its filter to route all the messages that have a Status field equal to BIC ERROR to the BICError terminal (rename it the match1 terminal to BICError).

All the rest of the messages with a Status field equal to BIC OK should be routed to the **default** terminal. See step 6 on page 565, where we created a similar node, for more detailed instructions.

Figure 8-118 shows the correct expression.

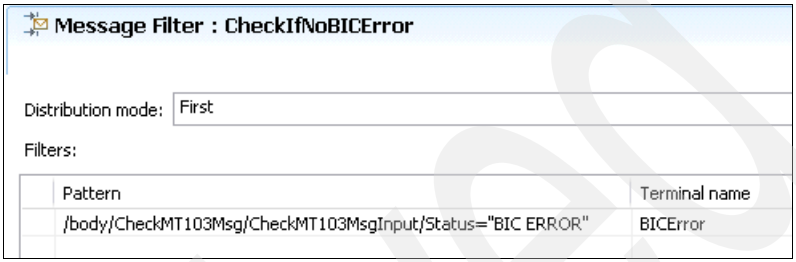

Figure 8-118 the filter expression in the BIC Error filter

b. Add the new business object map node called **PrepareOutput**. Wire it in the flow between the **BICError** output terminal of the **CheckIfNoBICError** node and the **in** terminal of the **Callout** of **RejectBIC** service.

Double-click the node to create a new map, and implement a simple BO map called `ConvertToOutput2`. Again use the namespace `http://www.ibm.com/Redbooks/ESB_Scenario1`. This node is almost the same as described in step 7 on page 566.

Drag the **CheckMT103MsgInput** element from the input to the **createInput** element on the output (Figure 8-119).

Figure 8-119 The mapping in the ConvertToOutput2 node.

c. Add a similar business object map node between the **default** terminal of the **CheckIfNoBICError** and the **in** terminal of the **Callout** of the service **ProcessValidMessage**. Create and save the business object map (name it `ConvertToObject3`). The node looks the same as the one in step b on page 577.

11. Save and close all editors.

12. Select **Project** → **Clean** → **Clean all projects**. You should not see any errors.

Your message flow is ready. Figure 8-120 shows the complete mediation.

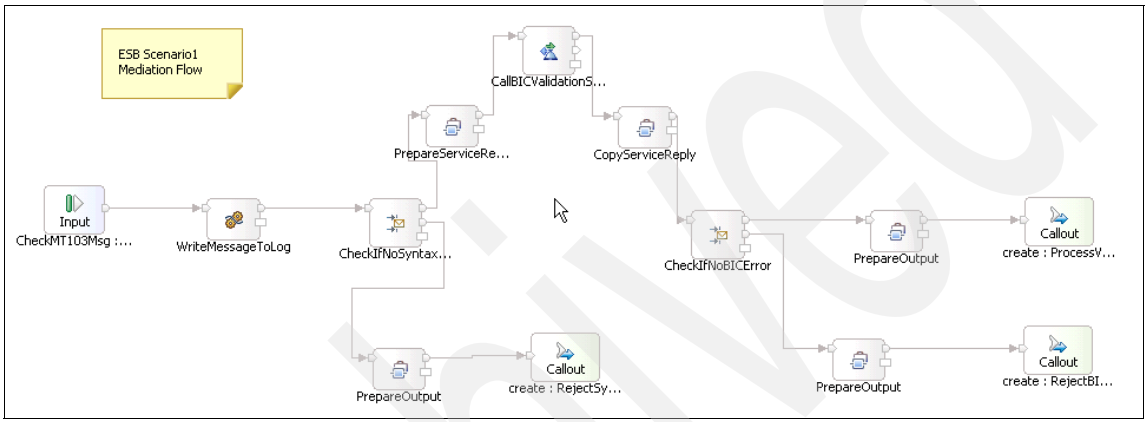

Figure 8-120 Completed mediation flow

8.4.6 Deploying and testing the scenario on WebSphere ESB

In this section, we test the whole scenario by using various test files. We place the files in the MT103_Input folder, where the Inbound Flat File Adapter picks it up. WebSphere ESB then runs the MT103MsgSWIFTToXML WebSphere Transformation Extender map to convert the data from its SWIFT format to the XML format.

We see the result of the transformation in the output log, displayed in the Console view. After checking the status of the message and checking the BIC validation results, the message is routed to one of three outbound adapters. The outbound adapter calls the WebSphere Transformation Extender MT103MsgXMLToSWIFT map to extract the original message from the XML document.

We prepared several test files that you can find in the additional materials that are available for download. For more information, see Appendix E, “Additional material” on page 749. Depending of the test file, you should see the output in one of the output directories:

- `C:\$Redbook\ESB_Scenario\BICReject`

  This directory contains the messages that have an invalid BIC.

- `C:\$Redbook\ESB_Scenario\SWIFT_Gateway`

  This directory contains the good messages.

- `C:\$Redbook\ESB_Scenario\SyntaxReject`

  This directory contains the messages that have invalid syntax.

To deploy the ESB_Scenario1 module to the WebSphere ESB server:

1. Open the **Servers** view.
2. Start WebSphere ESB Server v6.1 if it is not already started.
3. Add the **ESB_Scenario1App** and **ServicesApp** projects to the server.
4. Wait until both your applications have a Status of *Started*.
5. Go to the `C:\$Redbook\TestFiles\` directory and examine the contents of the different test files. Then copy and paste them to the `C:\$Redbook\ESB_Scenario\MT103Input` directory.

   The files should disappear and move with changed names to the different Output folders.

   Observe the Console view, where the whole Service Message object is logged. In the Body part of the Service Message object, you can find the MT103Msg business object, which is the result of the WebSphere Transformation Extender transformation.

Figure 8-121 shows the result of the transformation of the `MT103.1.syntNOK.txt` file. This file should be routed to the SyntaxReject subfolder.

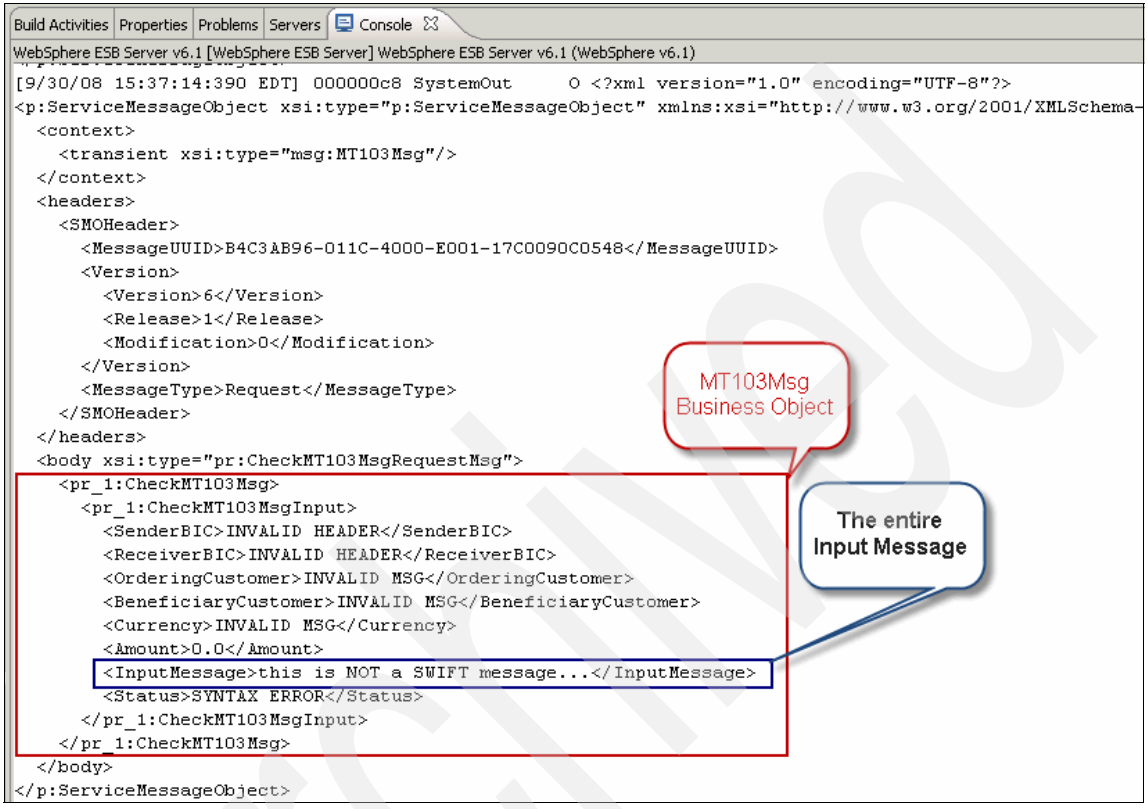

Figure 8-121 SystemOut log of WebSphere ESB Server v6.1

6. Repeat the same test for the other messages.

8.5 Building the WebSphere Message Broker Message scenario

Next we create the WebSphere Message Broker flow that interacts with WebSphere Transformation Extender for Integration Servers.

8.5.1 Roadmap for building the WebSphere Message Broker scenario

Figure 8-122 shows the roadmap to build the WebSphere Message Broker scenario. We place and configure two WTX Map nodes and a Collector node to finish the preconfigured flow.

This scenario uses two WebSphere Transformation Extender maps, the one created in the ESB scenario and one new map. You can choose either of the following options:

- Create the Transformation Extender maps following the instructions in 8.3.3, “Creating the WebSphere Transformation Extender maps” on page 496, through 8.4.3, “Creating the SWIFT to XML transformation map” on page 515, and 8.6, “Creating the second WebSphere Transformation Extender map” on page 582.
- Import the completed maps from the supplied Project Interchange file. See Appendix E, “Additional material” on page 749.

Then we deploy the solution to WebSphere Message Broker and test it.

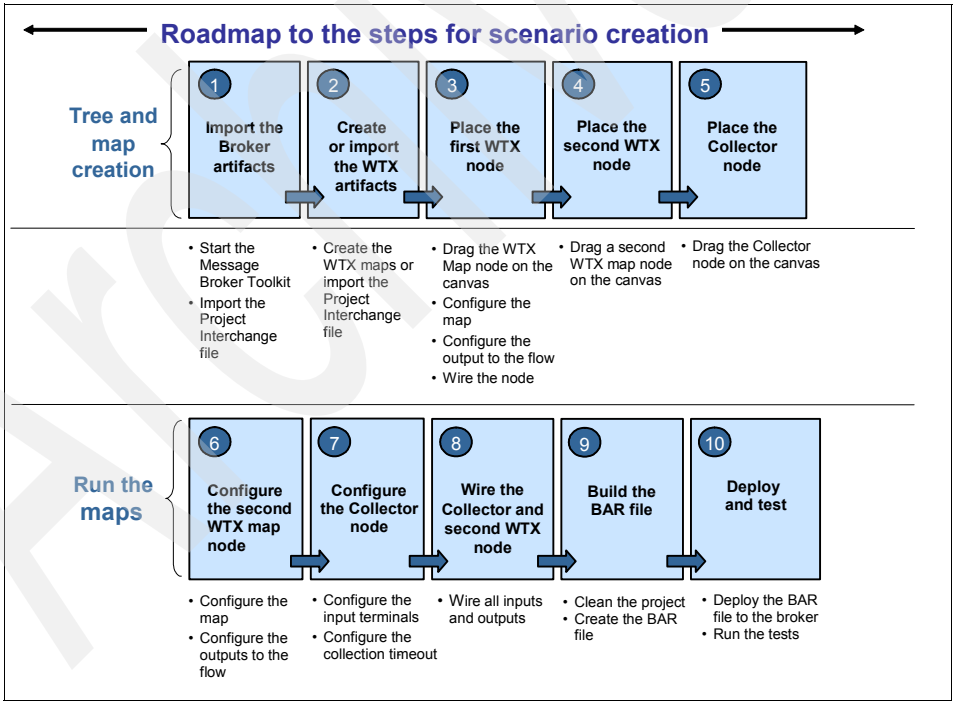

Figure 8-122 Roadmap for the WebSphere Message Broker scenario

8.6 Creating the second WebSphere Transformation Extender map

In the WebSphere Message Broker scenario, we use a second WebSphere Transformation Extender map. This second map routes the original MT103 message to a WebSphere MQ queue (SCENARIO2_SWIFT_GATEWAY) and generates a unique alert file based on business conditions. That is, we generate an alert file only if the amount of the transaction, converted in euro, is higher than €10,000 *or* this converted amount equals 0. Figure 8-123 shows the second map with its two inputs and two outputs.

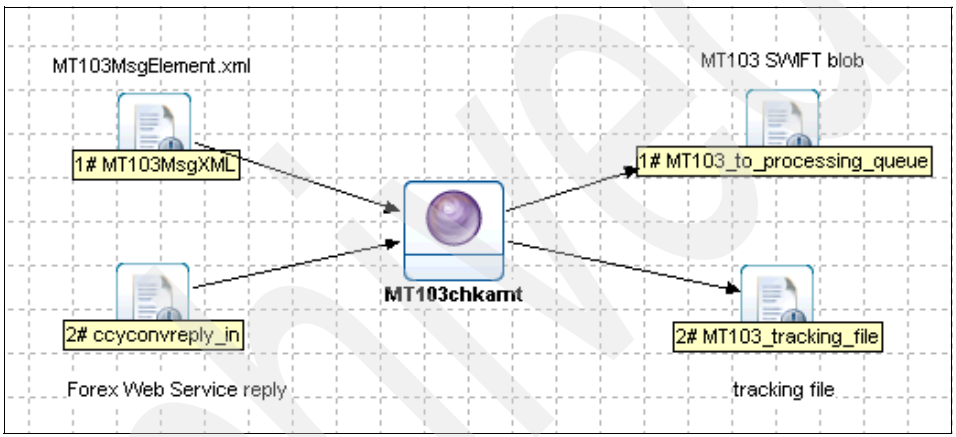

Figure 8-123 Overview of the MT103chkamt map

Similar to before, all input and output card settings are overridden when inserting this map into the WebSphere Message Broker flow. Therefore, we do not need to concerned about the settings that we are going to use. They are for map development purposes only.

To create this map:

1. Create a custom type tree for the alert file. You can create this in any format you choose. For this scenario, we started from an existing database table and used the Database Interface Designer to generate a type tree from a table. However, you can also create the type tree manually. Table 8-4 shows the metadata definition that we used.

Table 8-4 Alert file metadata table definition

Object	Class/subclass	Size	Range	Properties
DBTable	Group/implicit	--	1:1	--
Row	Group/explicit/delimited with pipe infix	--	s	Terminated with <CR><LF>
TRANS_ID	Item/number integer	01-20	0:1	
SENDER_BIC	Item/text	01-30	0:1	
RECEIVER_BIC	Item/text	01-30	0:1	
CURRENCY	Item/text	01-03	0:1	
AMOUNT	Item/number decimal	01-30	0:1	Decimal 0-4 digits
AMOUNT_EUR	Item/number decimal	01-30	0:1	Decimal 0-4 digits
STATUS	Item/text	01-50	0:1	
BIC_RETURN_CODE	Item/text	01-50	0:1	
BIC_RETURN_MESSAGE	Item/text	01-50	0:1	

Disassociating the metadata from the physical data: With WebSphere Transformation Extender, you can disassociate the metadata (type tree) from the physical data (adapter) and its location (adapter commands). Adapters deal with connectivity to the physical data, whether it is a source (GET) or a target (PUT), which provides the following advantages:

- Changing the version for a specific application (for example, migrating from WebSphere MQ 5.3 to WebSphere MQ 7 or Oracle 9i to 10g) is only a matter of changing the adapter. The map remains unchanged.
- As long as the data structure remains the same, you can get and put data basically anywhere you want without changing the mapping rules. In our scenario, the `MT103_TABLE.mtt` type tree has been built from a DB2 table, but we use the file target adapter that will be monitored by an application (for example Tivoli®). To see how to create a type tree from a table, see “Generating type trees” on page 308.

Figure 8-124 shows the type tree for the alert file.

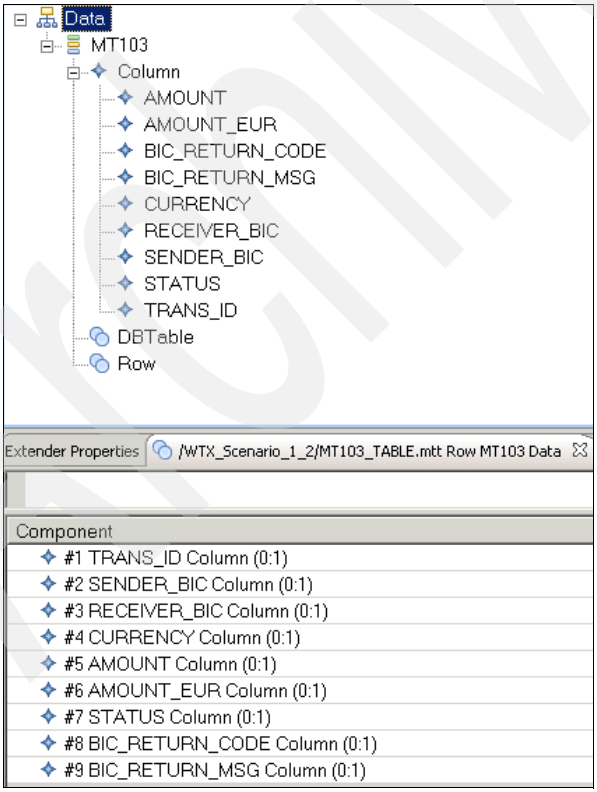

Figure 8-124 MT103 alert message type tree details

2. Create a new executable map by using one of the following methods:
   - From the menu bar, select **Map** → **New**.
   - Right-click **Outline/Composition** view root and select **New**.
   - Press Ctrl+M.
3. Name the map `MT103chkamt`.
4. Add the first input card, which reads the MT103 XML input. We were unable to create this card from scratch, but there is an easier way. Since we already used this input card in the previous map, we can simply copy it:
   a. From the MT103MsgSWIFTToXML map, right-click output card #1 **MT103MsgXML** and select **Copy**.
   b. In the Copy Card window (Figure 8-125), for Map, type `MT103chkamt` as the destination and for Copy Card As, select **Input.** There is no need to change the card name because we are copying it to another map.

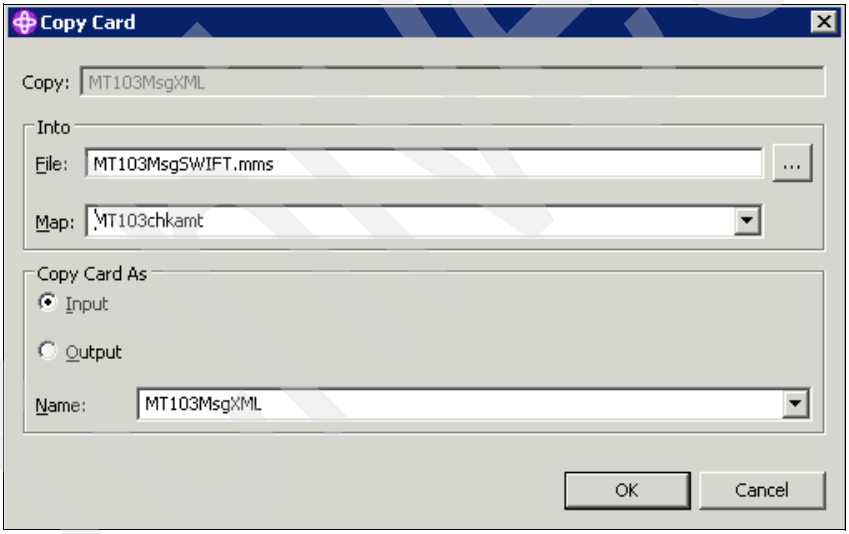

Figure 8-125 Copying the MT103MsgSWIFTToXML output card of as the input card for MT103chkamt

The card is now added as an input card of the MT103chkamt map. The rule column has been erased because this is an input card.

5. Create the second input card. The WebSphere Message Broker flow supplies this card with the reply from the currency conversion Web service, which has been stripped from its SOAP envelope. Therefore import the schema that represents the reply message of this Web service.

   a. Select **Schema Files** → **Import** → **File system**.

   b. In the Import - File system window (Figure 8-126), select the source directory, **CurrencyConversionReply.xsd**, and the destination folder, which is **WTX_Scenario_1_2**. Then click **Finish**.

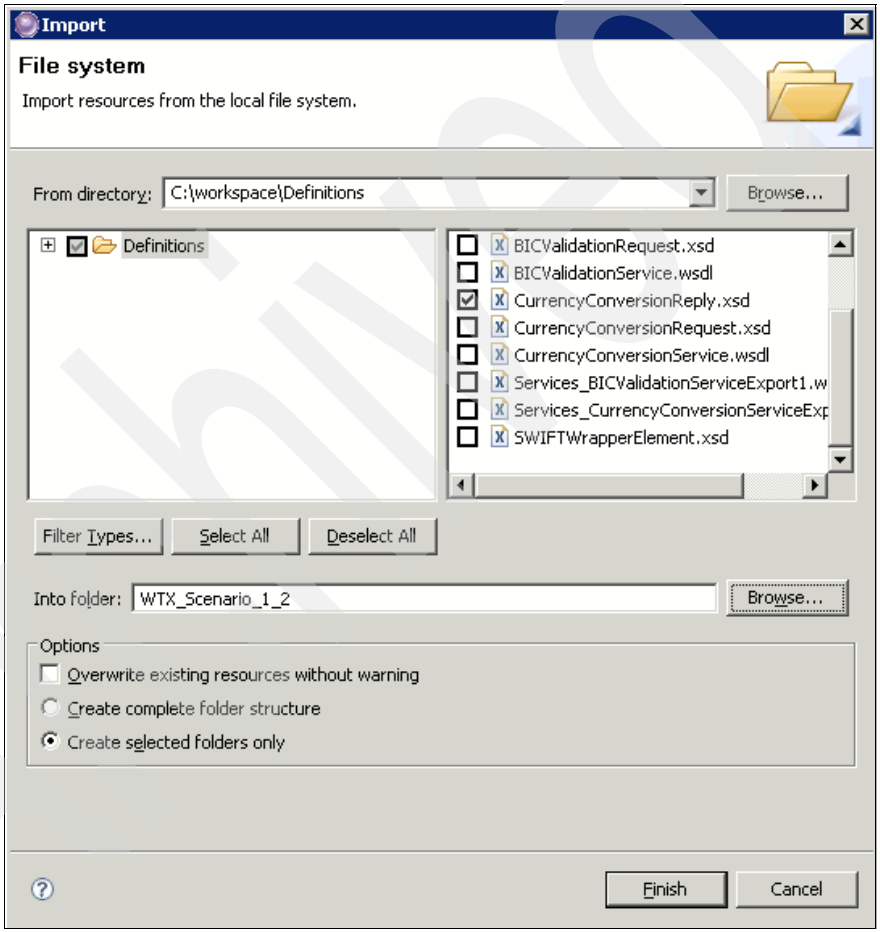

Figure 8-126 Importing the CurrencyConversionReply schema from the Web service definition

c. Create a new input card. Use this schema as the type tree setting and type the card at the XSD root itself.

d. Use an example of the Web service response as the source of the card by using the File adapter (Example 8-3).

Example 8-3 Web service CurrencyConversion Reply

```
<wtx:ConvertResponse
xmlns:wtx="http://www.ibm.com/Redbooks/CurrencyConversion">
<Reply>
<FromCurrencyCode>USD</FromCurrencyCode>
<ToCurrencyCode>EUR</ToCurrencyCode>
<BaseAmount>210.5</BaseAmount>
<ConvertedAmount>148.89</ConvertedAmount>
<ConversionRate>0.70732</ConversionRate>
</Reply>
</wtx:ConvertResponse>
```

6. Create the first output card. This card routes the whole MT103 message to the WebSphere MQ queue SCENARIO2_SWIFT_GATEWAY. You can put the whole message into a BLOB item, such as the OtherMSG element of the `MT103.mtt` type tree:

   a. Add an output card and name it `MT103_to_processing_queue`.

   b. Type this card by using `OtherMSG` from `MT103.mtt`.

   c. Use the File adapter with any path and file name. Remember this is just for testing, because it will be overridden when integrated in WebSphere Message Broker.

   d. Using the TEXT function, retrieve only the MT103 message from the input XML segment <InputMessage>. Again, use the NativeXML input schema, and therefore, the TEXT function is not necessary. Use the following rule for the MT103_To_Processing_queue card:

   ```
   TEXT (InputMessage:sequence:MT103Msg:global:MT103MsgXML)
   ```

7. Create the second output card, which generates the alert message and uses the type tree generated in step 1 on page 583:

   a. Add an output card named `MT103_tracking_file` to the map.

   b. Type this card by using `DBTable MT103 Data` from `MT103_TABLE.mtt`.

   c. Use the File adapter with any path and file name. Again this is for testing purposes. It will be overridden when integrated in WebSphere Message Broker.

d. Use a conditional functional map call to generate the alert based on a logical condition. The functional map should be called to generate data only in either of the following situations:

   - The amount converted in euros is higher than €10,000.
   - The amount converted in euros is equal to 0.

We use an IF function to test the value of ConvertedAmount of the Web service reply. If the condition is *true*, we call a functional map F_alert by using two arguments:

– The XML message itself
– The converted amount as returned by the Web service

This is a call/no call condition, because there is no `Else` argument.

Conditional function map: Conditional function map calls are a major feature of WebSphere Transformation Extender. We can add efficient logic to any kind of transformation process. Possibilities that this type of mapping implies are wide. We can make successive calls, iterative calls, calls based on multiple conditions, calls of various functional maps depending on a qualifier, and so on. Remember that a functional map is called for each occurrence of each argument (cartesian product of the arguments). If any argument is evaluated to NONE (or 0), the functional map is *not* called. Likewise, if you incorrectly type one of the arguments (that is, the record level instead of the file level) of the functional map call, it is called too many times, generating too many output data records.

In a pure WebSphere Transformation Extender scenario (without WebSphere Message Broker), you might alter the card default properties to generate data for output card #2 *only if* the functional map is called. That is to say only if there is data to generate. To do so, you can select **Transaction** → **OnSuccess** and set OnSuccess to **CreateOnContent** (Figure 8-127 on page 589). In a WebSphere Message Broker scenario, settings of the cards are overridden. Therefore, we must implement this logic in the broker itself (see Figure 8-7 on page 485).

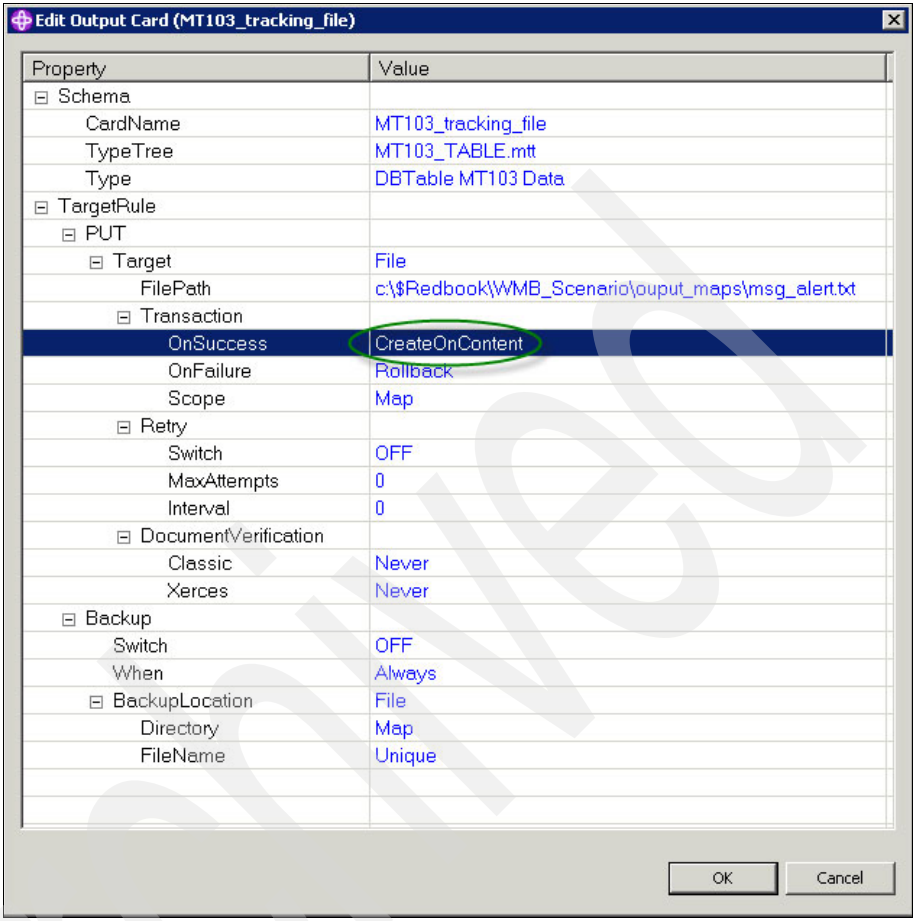

Figure 8-127 Changing the default OnSuccess setting to CreateOnContent

8. Create the functional map. We can create it manually, but it is better to use the Functional Map Wizard. First enter the map rule in the target field of the output card. Figure 8-128 shows the mapping rule for the Row (s) object of the MT103_tracking_file.

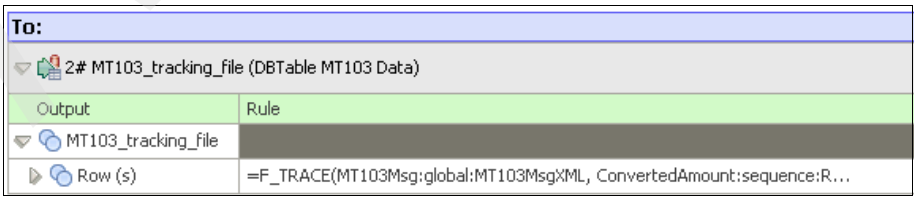

Figure 8-128 Mapping rule for the functional map

Enter the mapping rule as shown in Figure 8-129. Remember not to type the arguments, but drag the arguments in place.

```
=F_TRACE(MT103Msg:global:MT103MsgXML,
ConvertedAmount:sequence:Reply:sequence:ConvertResponse:global:
convreply_in)
```

Figure 8-129 Mapping rule for the functional map

9. After the functional map call is done, either click the **Functional Map Wizard** icon, or right-click and select **Functional Map Wizard**.
10. In the Functional Map Wizard window (Figure 8-130), in which you see the functional maps that it creates, the input cards for each argument, and the output card (one for each functional map), edit each card name:
    a. Highlight the card and click **Edit**.
    b. Change the card to a relevant name (inset in Figure 8-130), which makes things easier if you want to debug the map.
    c. After every card name is changed, back in the Functional Map Wizard, click **Create** to automatically generate the functional map or maps.

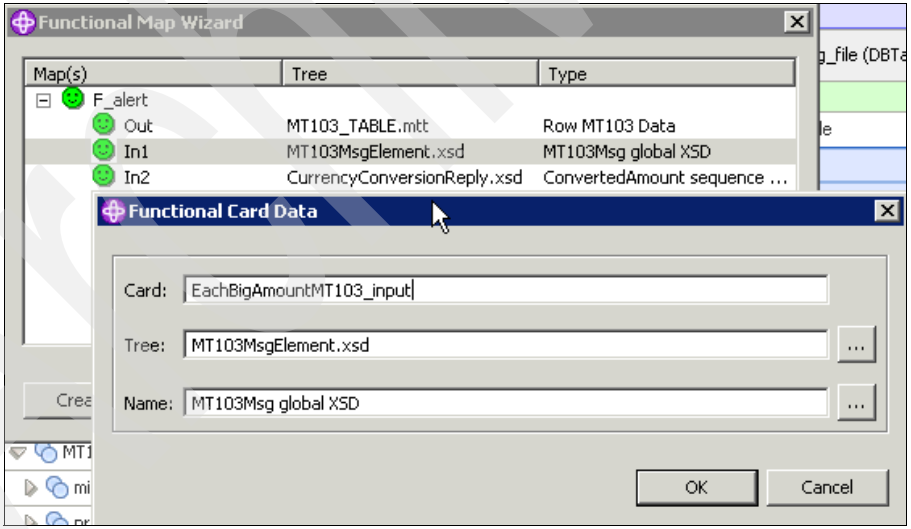

Figure 8-130 Renaming each card before creating the functional map

11. Map the elements of the functional map. Note that, in the Composition view of the navigator, you can see the dependency between the executable map and the functional map.

    a. Double-click the **F_alert** functional map to start mapping.

    b. Leave the first field (TRANS_ID) empty. It might be incremented by the target application.

       The STATUS output item has the mapping rule shown in Figure 8-131.

```
=IF (ConvertedAmount>10000, "TREASURY MANAGER APPROVAL NEEDED",
  IF (ConvertedAmount=0, "FOREX CONVERSION FAILED",
  Status:sequence:EachBigAmountMT103_input))
```

Figure 8-131 Mapping rule for the Status field

       This means that you get the Status input message field (OK or SYNTAX ERROR), unless there is an alert on the ConvertedAmount from Input Card#2, meaning that this amount equals 0 or is higher than €10,000.

    c. Map all other fields from input card #1.

12. Build and run the map. You can change the input card #2 data `convertresponse.xml <ConvertedAmount>` segment value to test that the logic is working properly (Figure 8-132).

```
<wtx:ConvertResponse xmlns:wtx="http://www.ibm.com/Redbooks/CurrencyConversio
<Reply>
<FromCurrencyCode>USD</FromCurrencyCode>
<ToCurrencyCode>EUR</ToCurrencyCode>
<BaseAmount>210.5</BaseAmount>
<ConvertedAmount>148.89</ConvertedAmount>
<ConversionRate>0.70732</ConversionRate>
</Reply>
</wtx:ConvertResponse>
```

Figure 8-132 Changing the ConvertedAmount of the convertresponse.xml file for testing

8.6.1 Building the WebSphere Message Broker scenario

Now we import the artifacts that we predefined and configure the message flow so that it calls the WebSphere Transformation Extender maps. We assume that you started from an empty workspace and created the WebSphere Message Broker default configuration.

Importing the prepared WebSphere Message Broker artifacts into the WebSphere Message Broker Toolkit

To import the WebSphere Message Broker artifacts into the tooling environment:

1. Start the Message Broker Toolkit and open the Broker Application Development perspective.
2. Select **File** → **Import** → **Other** → **Project Intechange**.
3. In the window that opens, click **Next**.
4. In the next window:
   a. Browse to the location where you stored the `Scenario2_start.zip` file.
   b. Select **Scenario2_start.zip** file and click **Open**.
   c. In the previous window, click **Select All**.
   d. Click **Finish**.

You now see the projects listed (Figure 8-133) that have been imported into your Message Broker Toolkit.

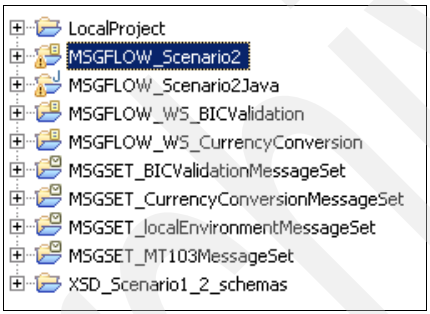

Figure 8-133 The imported projects

Optional: Importing the WebSphere Transformation Extender project into the development environment

It is possible that you did not complete all steps in 8.3.3, “Creating the WebSphere Transformation Extender maps” on page 496, through 8.4.3, “Creating the SWIFT to XML transformation map” on page 515, and 8.6, “Creating the second WebSphere Transformation Extender map” on page 582. In this case, you can now import the project that contains the completed WebSphere Transformation Extender maps into the tooling environment. See Appendix E, “Additional material” on page 749, for details about the available files.

To import the WebSphere Transformation Extender project into the development environment:

1. Select **File** → **Import** → **Other** → **Project Intechange**.
2. In the window that opens, click **Next**.
3. In the next window:
   a. Browse to the location where you stored the `WTX_Scenario1_2.zip` file.
   b. Select the **WTX_Scenario1_2.zip** file and click **Open**.
   c. Select the **WTX_Scenario_1_2** check box and click **Finish**.

You now have these projects in your workspace (Figure 8-134).

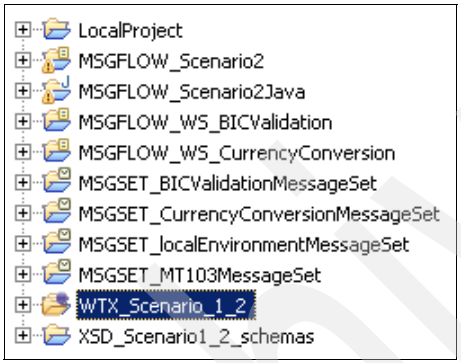

Figure 8-134 The projects, including the WebSphere Transformation Extender project

Opening the message flow and replacing the Passthru nodes

We now replace the Passthru nodes in the message flow by the WTX Map nodes and a Collector node:

1. In the Broker Application Development perspective (Figure 8-135), navigate to and double-click the **Scenario2.msgflow** file.

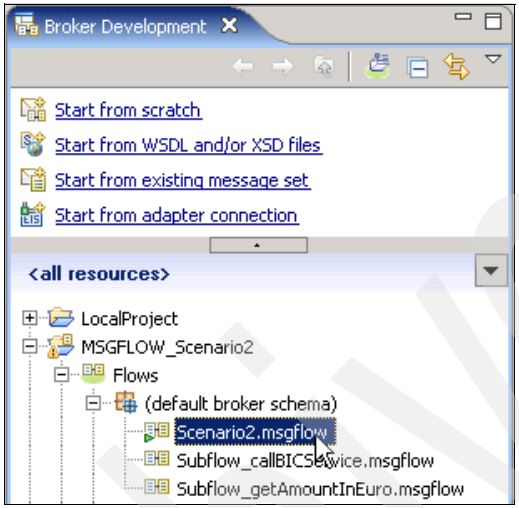

Figure 8-135 Navigating to the Scenario2.msgflow file

Figure 8-136 shows the nodes of the prebuilt message flow.

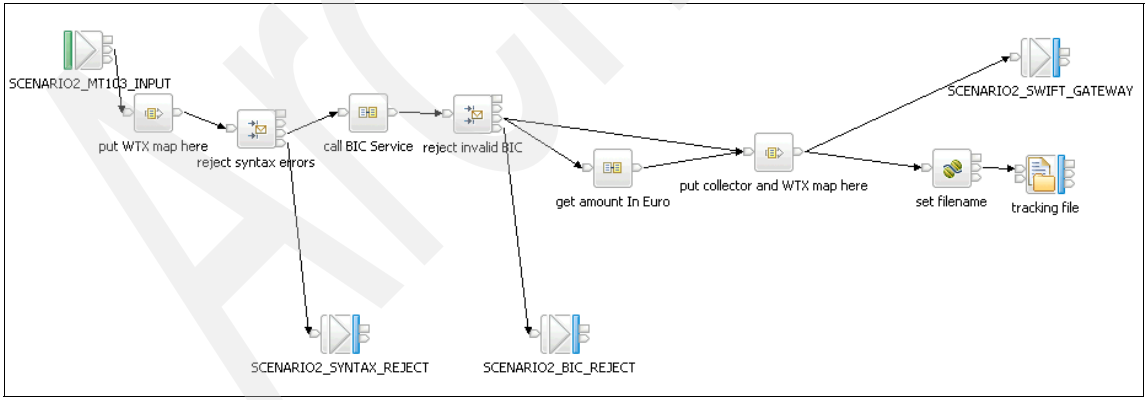

Figure 8-136 The prebuilt message flow

2. As shown in Figure 8-137, right-click the **put WTX map here** node and select **Delete**.

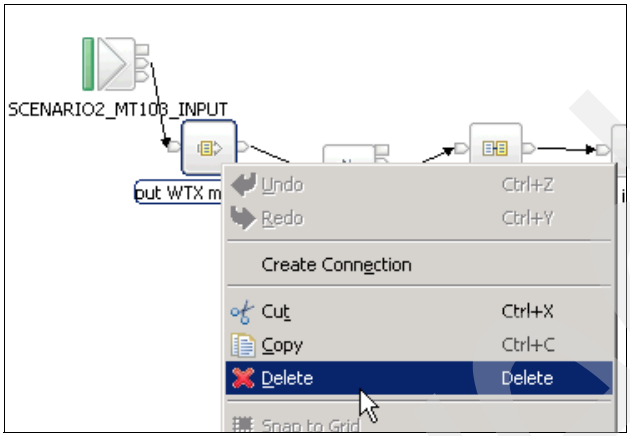

Figure 8-137 Deleting the put WTX map here node

3. Rename a WTX Map node:
   a. Select a WTX Map node from the WebSphere TX folder in the Palette.
   b. Drag the node to the canvas where you just deleted the previous node.
   c. Right-click the node and select **Rename**.
   d. Name the node `analyze SWIFT` (Figure 8-138). Notice that the only output terminal available on the node is the Failure terminal. This changes as we configure the node.

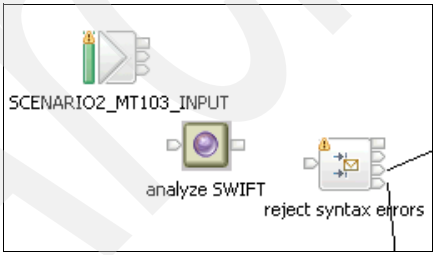

Figure 8-138 Adding the WTX Map node to the canvas

4. Point the node to the WTX map:
   a. Click the **analyze SWIFT** node.
   b. In the Properties window (Figure 8-139), for Executable map, click **Browse**.

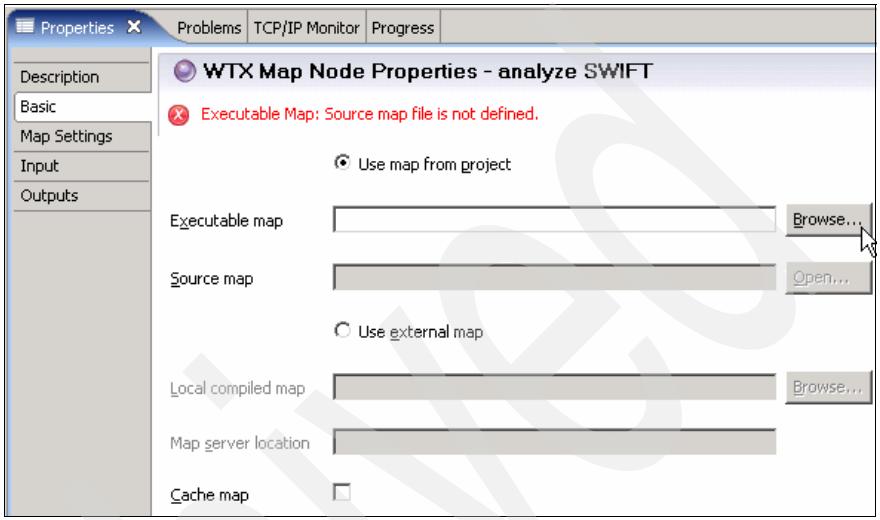

Figure 8-139 WTX Map Node Properties - Browsing to the executable map

c. Select the **MT103MsgSWIFTToXML** map (Figure 8-140) and click **OK**.

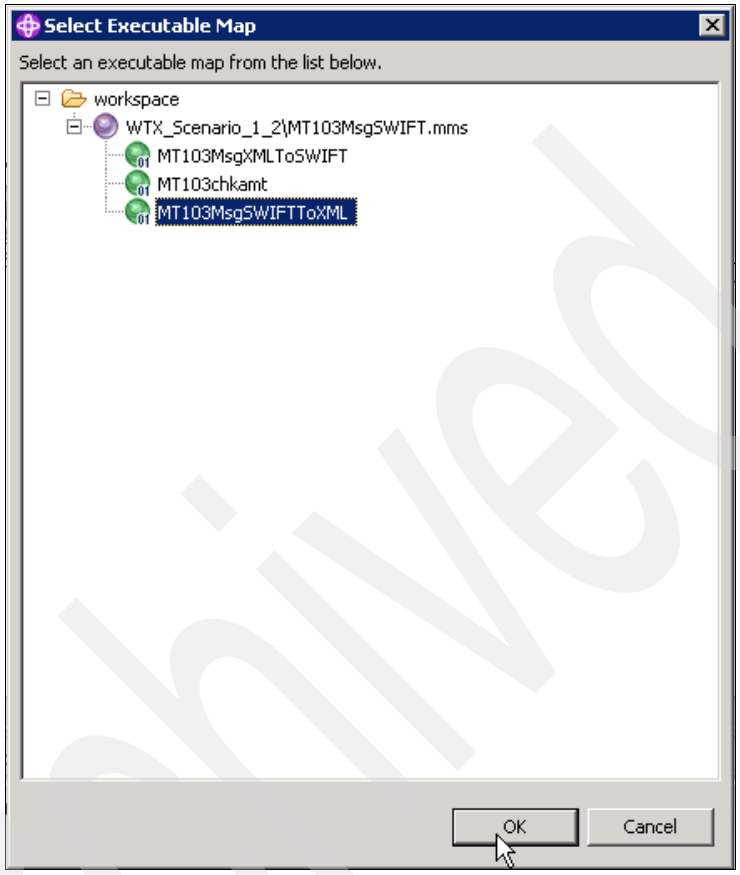

Figure 8-140 Selecting the executable map

Now you see that both fields for Executable map and Source map are filled.

d. Click the **Cache map** check box to ensure that the map stays loaded in memory for subsequent messages passing through the message flow. This optimizes performance. Figure 8-141 shows the completed window.

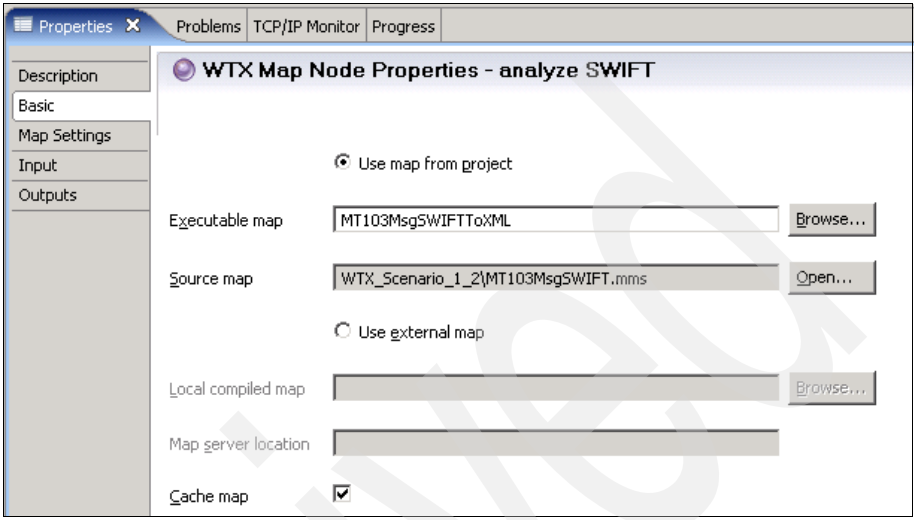

Figure 8-141 The map selected

5. Configure the node to route the data of output card #1 back to the message flow and define for the node how the data will be represented in the message flow:

   a. Click the **Outputs** tab and click **Add** (Figure 8-142).

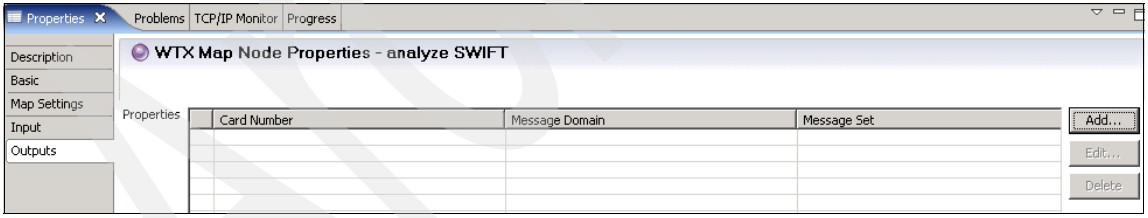

Figure 8-142 Adding the output card

b. In the Edit Properties entry window (Figure 8-143), enter the following information for one card:

- For Card Number, type `1`.
- For Message Domain, type `XMLNSC`.
- For Coded Char Set ID, type `1208`.

With these settings, you indicate that you want to route the first output card back to the message flow, have the message flow parse the data in the XMLNSC domain, and represent the data as Unicode.

Click **OK**.

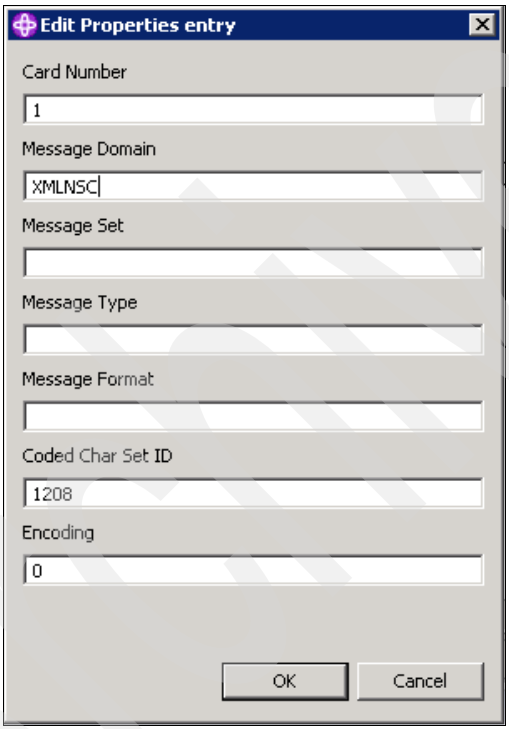

Figure 8-143 Output card properties

A new output terminal is available on the node as shown in Figure 8-144. The default name generated for the output terminal is `out1`.

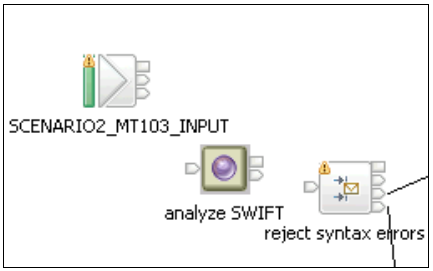

Figure 8-144 The WTX node with the new terminal

6. Wire the new node:
   a. Wire the **out** terminal of the **SCENARIO2_MT103_INPUT** node to the **in** terminal of **WTX Map node**.
   b. Wire the **out1** node of the **WTX Map** node to the **In** terminal of the **reject syntax errors node**.

      Important: Use care in wiring the terminals so that you do not mix up the terminals. To verify that you have wired the terminals correctly, hover your cursor over a wire. A message box opens that shows which terminals are connected by that wire.

   c. Save the flow by clicking **File** → **Save**.

7. Replace the second Passthru node:

   a. Right-click the **put collector and WTX map here** node and select **Delete** (Figure 8-145).

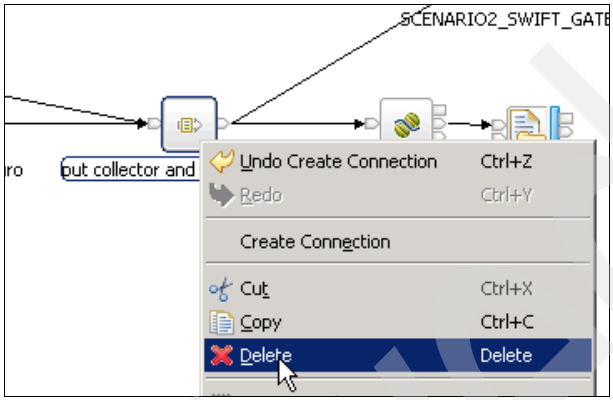

Figure 8-145 Deleting the second Passthru node

   b. Add a Collector node and a WTX Map node where you deleted the Passthru node. You can find the Collector node in the Routing drawer of the Palette.

   c. Rename the WTX Map node to `propagate SWIFT` (Figure 8-146).

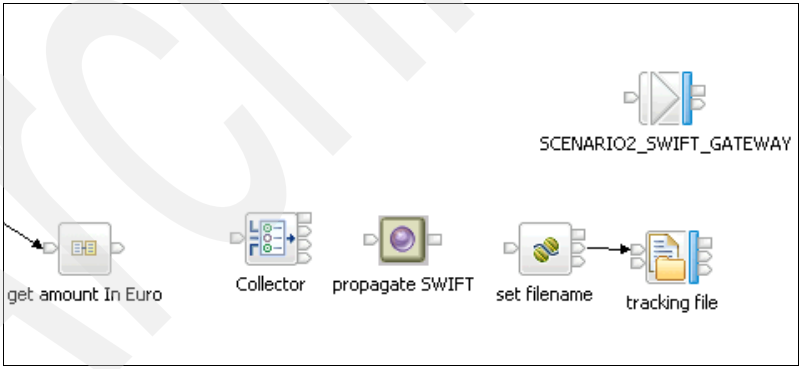

Figure 8-146 The Collector node and the WTX node added

8. Click the new **propagate SWIFT** node and set the properties:
   a. On the **Basic** tab, for Executable map, click **Browse**.
   b. In the Select Executable Map window (Figure 8-147), navigate to **MT103chkamt** and click **OK**.

Figure 8-147 Selecting the map for the second WTX Map node

c. Back on the **Basic** tab of the Properties window (Figure 8-148), select the Cache map check box to keep the map in memory and to help with performance.

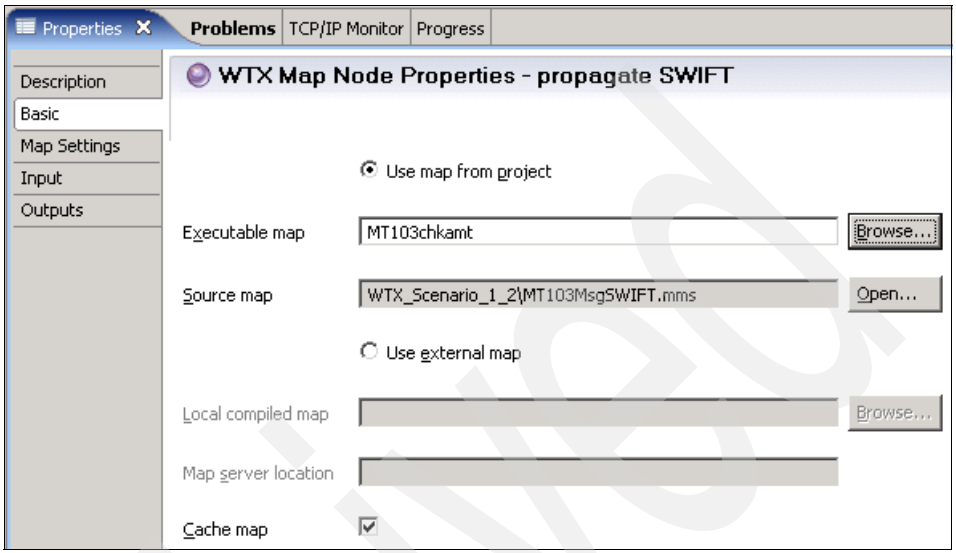

Figure 8-148 Basic configuration of the second WTX Map node

9. Configure the output cards for which you want the data to be sent back into the message flow:

   a. In the Properties window (Figure 8-149), select the **Outputs** tab and click **Add**.

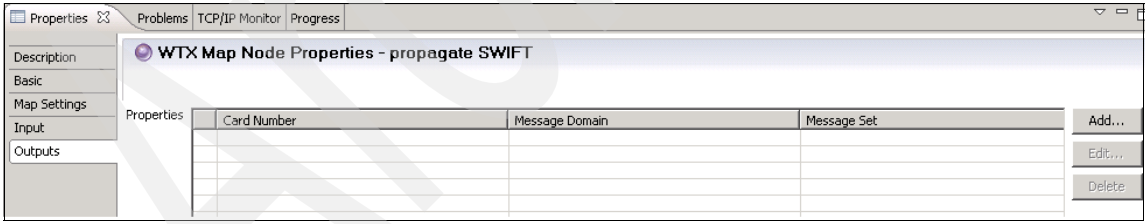

Figure 8-149 Adding outputs

b. In the Add Properties entry window (Figure 8-150), complete the properties for the first output card by entering the following information:

- For Card Number, type `1`.
- For Message Domain, type `BLOB`.

Then click **OK**.

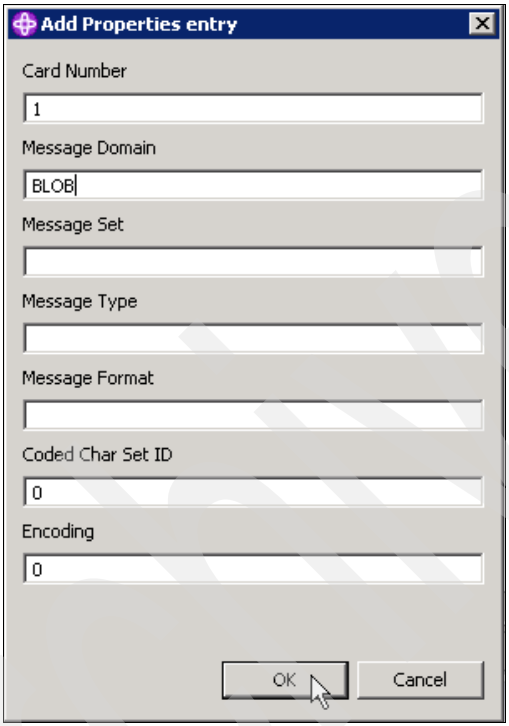

Figure 8-150 Output card 1

c. Repeat steps a and b for the second output card. In the Add Properties entry window, enter the following information and click **OK**:

- For Card Number, type `2`.
- For Message Domain, type `BLOB`.

Figure 8-151 shows the **Outputs** tab.

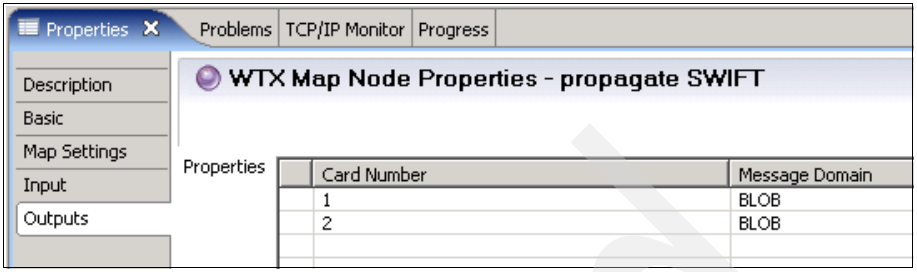

Figure 8-151 Outputs tab with two cards

The node is now configured to route the data of the WebSphere Transformation Extender map's output cards 1 and 2 back to the message flow. The new node has received two new output terminals. (It received them from the moment you configured the executable map). Figure 8-152 shows how the node should look.

Figure 8-152 The WTX Map node with the new terminals

10. Create the input terminals of the Collector node.

Collector node: The Collector node is the key to having a WebSphere Transformation Extender map with multiple input cards in a message flow. We give the Collector node an input terminal for each input card that we want to feed from WebSphere Message Broker. The Collector node will then assemble the data received on these input terminals and group them in a collection. The WTX Map node receives this collection and feeds the different parts of it into the right input cards.

To identify the input card that needs to consume the data received on an input terminal of the Collector node, you must follow this naming convention: *The name of the input terminal on the Collector node must exactly match the name of the input card of the map referred to in the WTX Map node.* We illustrate this naming convention in the steps that follow.

a. Verify the input card names.

   i. Switch to the **Transformation Extender** perspective.

   ii. Go to the **WTX_Scenario1_2** project.

   iii. Double-click the **MT103MsgSWIFT.mms** map source file.

   iv. In the Outline view, double-check the names of the input cards for the **MT103chkamt** map (Figure 8-153).

   These are the exact names that we must use as input terminals of the Collector node. The names are case sensitive.

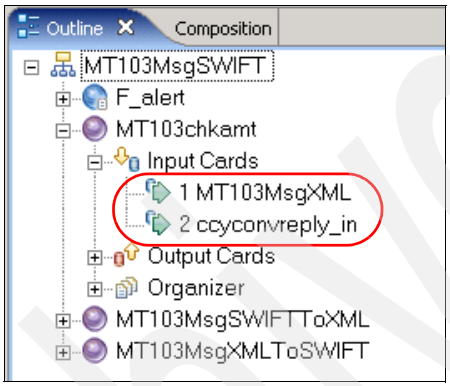

Figure 8-153 Checking the names of the input cards

b. Create the input terminals:

   i. Return to the **Message Broker Development** perspective.

   ii. Right-click the **Collector node** and select **Add Input Terminal** (Figure 8-154).

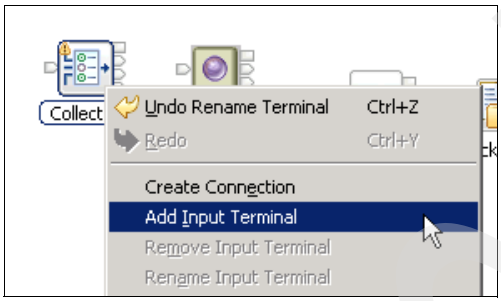

Figure 8-154 Adding an input terminal

   iii. Enter the name `MT103MsgXML` (Figure 8-155) and click **OK**.

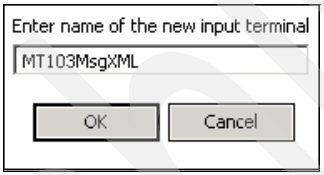

Figure 8-155 Setting the name of the first input terminal

   iv. Repeat these steps exactly the same for the second input terminal. However, this time, enter the name `ccyconvreply_in` (Figure 8-156).

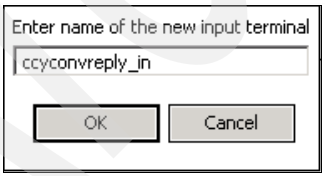

Figure 8-156 Setting the name of the second input terminal

11. Set the Collection expiry.

> **Collection expiry:** The Collection expiry indicates the maximum time interval between the arrival of both input triggers at the Collector node. A timer is started when the first input message enters the Collector. If the second input does not arrive within the Collection expiry interval, the collection expires and is sent to the Expire terminal. Always set a Collection expiry to avoid waiting endlessly for non-arriving triggers.

a. Click the **Collector node**.

b. Click the **Basic** tab of the Properties window (Figure 8-157). Notice that the MT103MsgXML and ccyconvreply_in input terminals have been added. In the Collection expiry field, enter an interval in seconds, for example, 60.

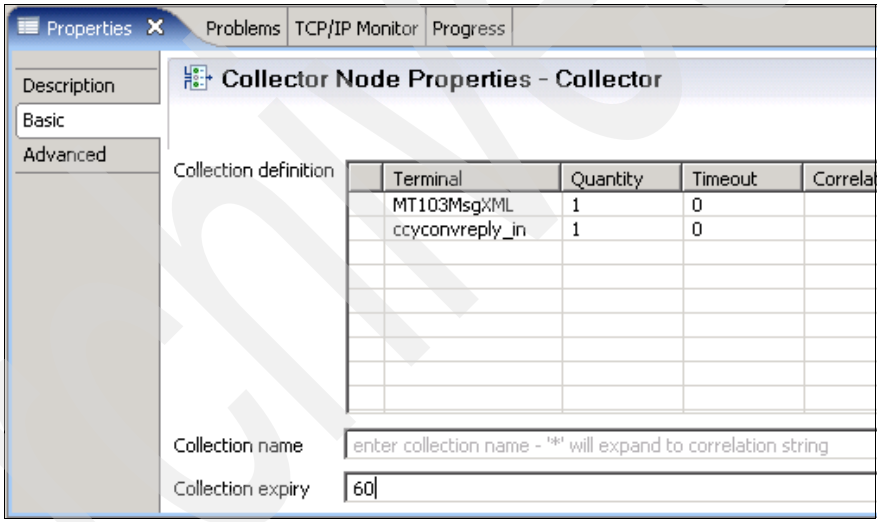

Figure 8-157 Collection expiry

12. Wire the nodes. Table 8-5 shows which terminals to connect.

Table 8-5 Wiring chart

From node	From terminal	To node	To terminal
reject invalid BIC	BIC_OK	Collector	MT103MsgXML
reject invalid BIC	BIC_OK	get amount in euro	Input
get amount in euro	Output	Collector	ccyconvreply_in
Collector	Out	WTX Map1	In
Collector	Expire	WTX Map1	In
WTX Map1	out1	SCENARIO2_SWIFT_GATEWAY	In
WTXMap1	out2	set filename	In

Note: We wired both the Collector node's out and Expire terminals to the WTX Map1 node's in terminal. In this case, if an expiry is incomplete, it is still sent to the WTX map. The map results in return code 12, "`Source not available`."

If you do not want this return code, wire the Expire terminal of the Collector node to other nodes that implement the required behavior because not all inputs arrive in a reasonable timeframe.

Figure 8-158 shows the flow after wiring the terminals.

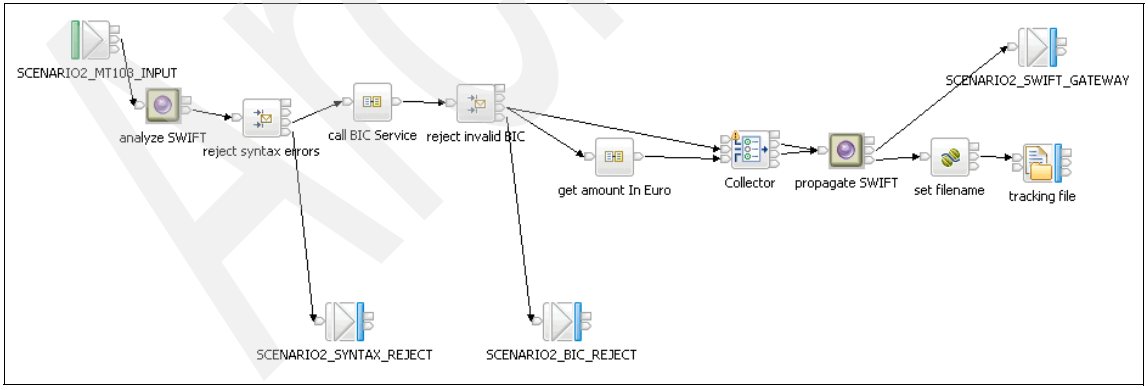

Figure 8-158 Completed message flow

13. Save your flow by pressing Ctrl+S.

14. Clean your projects:

    a. Select **Project** → **Clean**.
    b. In the Clean window (Figure 8-159), select **Clean all projects** and click **OK**.

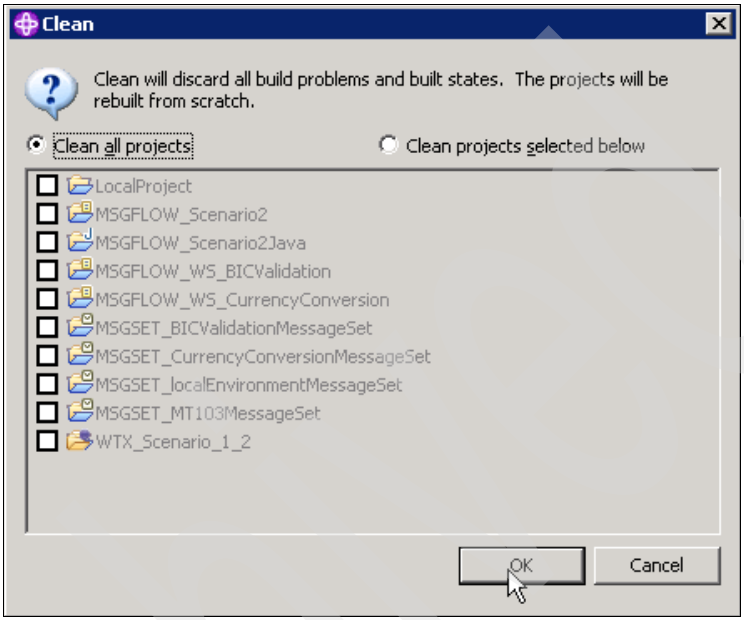

Figure 8-159 Cleaning the projects

Note: Although we have now built the message flow, the flow is not ready to use in a production environment. We built this flow for the purpose of demonstrating the integration of WebSphere Transformation Extender and WebSphere Message Broker. We did not implement any error handling routine in WebSphere Message Broker.

If you want a production-ready message flow, consider the type of error handling that you need and the following tips:

- Connect the catch terminal of the input node to an error handling routine.
- Connect the catch terminal of the collector node to error handling nodes. If you do not do this, and an exception is thrown downstream of the collector node, the node continuously retries to process the same collection.

For general design guidelines around WebSphere Message Broker flows, see the WebSphere Message Broker documentation.

8.6.2 Deploying and testing the scenario on WebSphere Message Broker

In this section, we guide you though the deployment and test of the flow.

Deploying the message flow

Deploy and test the WebSphere Message Broker flow:

1. Switch to the **Broker Administration** perspective.
2. In the Broker Administration Navigator, right-click **Broker Archives** and select **New** → **Message Broker Archive** (Figure 8-160).

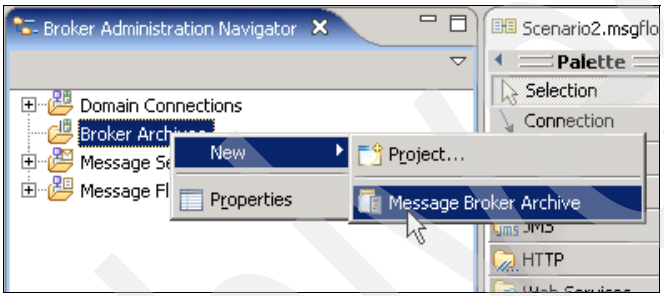

Figure 8-160 Creating a new BAR file

3. In the New Message Broker Archive window (Figure 8-161), for Project, select **MSGFLOW_Scenario2** and for Name, type `Scenario2`. Then click **Finish**.

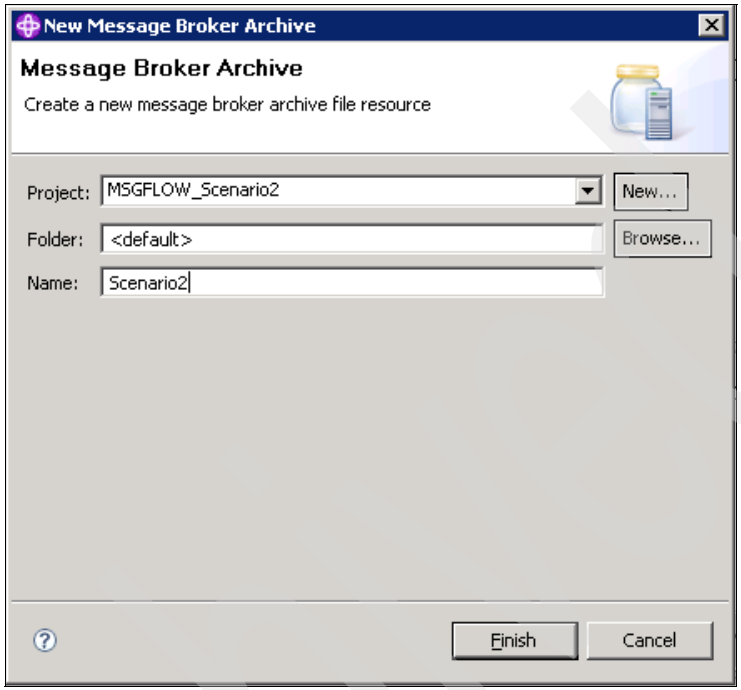

Figure 8-161 Creating the BAR file

4. In the BAR file editor, on the **Prepare** tab (Figure 8-162), select the Message Flows, the Message Sets, and the Java projects.

> **Map archive (MAR) check box:** Notice the MAR check box in the BAR file editor. A MAR is a compressed file that contains the compiled versions of a WebSphere Transformation Extender map for Windows, UNIX, and z/OS platforms. This MAR file is packaged in the BAR file and sent to the broker.
>
> If you have existing MAR files, add them to your BAR file here. If you do not have MAR files, they are created and automatically added when you build the BAR file.

Click the **Build Broker archive** button.

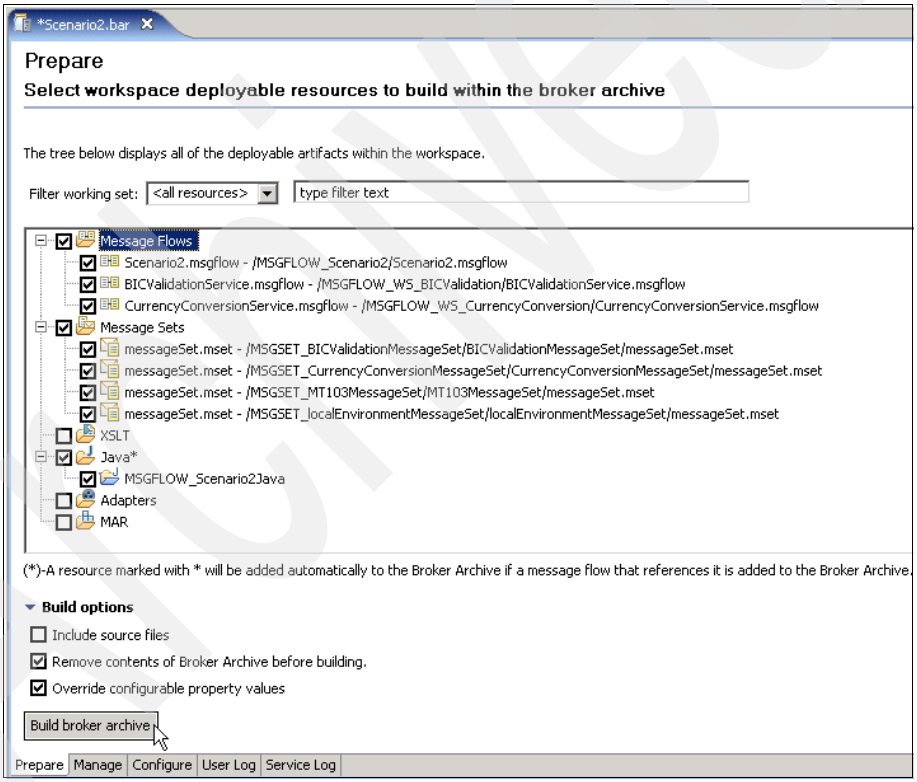

Figure 8-162 Selecting the artifacts

5. If you see the Override configurable properties window (Figure 8-163), click **OK**.

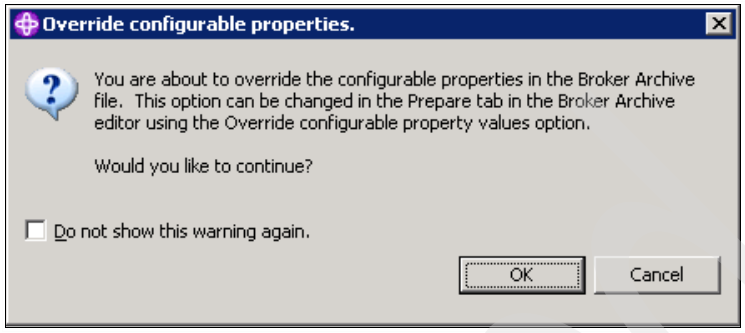

Figure 8-163 Override configurable properties window

6. When you see the message window (Figure 8-164) indicating that the operation completed successfully, click **OK**.

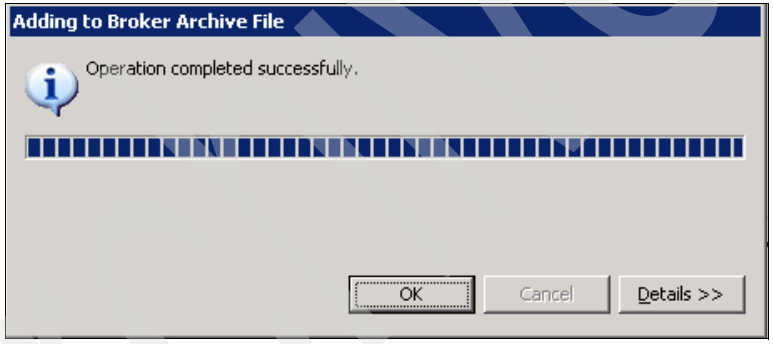

Figure 8-164 Build complete

7. Save the BAR file by pressing Ctrl+S.

8. Deploy the BAR file:
   a. Right-click **Scenario2.bar** and select **Deploy File** (Figure 8-165).

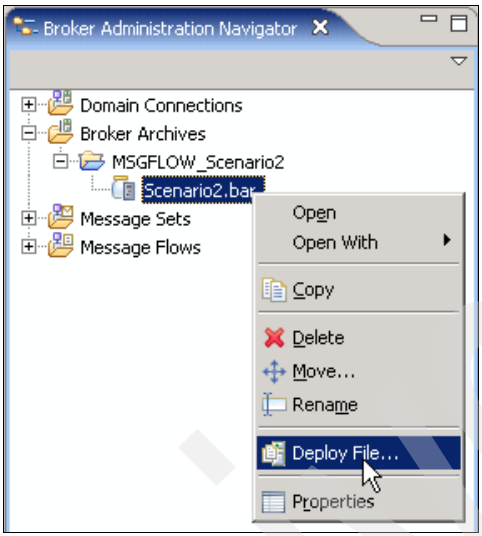

Figure 8-165 Deploying the BAR file

b. In the Deploy a BAR File window (Figure 8-166), select the default execution group and click **OK**.

Figure 8-166 Deploying the BAR file

Figure 8-167 shows all the artifacts deployed to the default execution group.

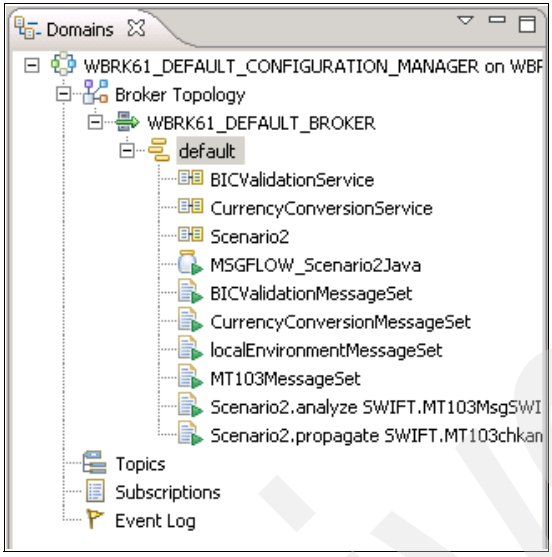

Figure 8-167 The artifacts deployed

Inspecting the artifacts in the Message Flow project

To test the message flow, we use the WebSphere Message Broker Unit Test Client, which has been available since version 6.1.

Switch back to the **Broker Application Development** perspective and view the contents of the **MSGFLOW_Scenario2** project.

Figure 8-168 shows the contents of the project.

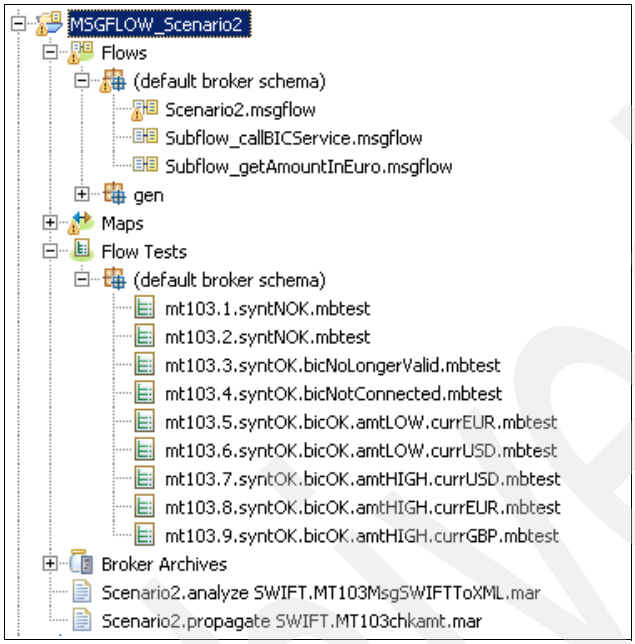

Figure 8-168 Contents of the MSGFLOW_Scenario2 project

The project shows the following different artifacts:

- The message flow and the subflows
- The msgmap files
- Flow test files

  Double-clicking any of these files launches the Unit Test Client and sets it all ready to launch a test.

- The Broker archive file for deployment
- The Map archive files that were generated

  These map archives are added in the Broker archive.

There are nine predefined test files divided in four categories. The first two tests fail because of invalid syntax:

- `mt103.1.syntNOK.mbtest`

  This `mt103.1.syntNOK.mbtest` test is where we send a message that has nothing to do with SWIFT to the input queue. This should be discovered by the first map and a message should be created on the SCENARIO2_SYNTAX_REJECT queue.

- `mt103.2.syntNOK.mbtest`

  In the `mt103.2.syntNOK.mbtest` test, we use a SWIFT message, but we deliberately malformed one of the amounts. Similar to the previous test, this qualifies as bad syntax and a message should be sent to the SCENARIO2_SYNTAX_REJECT queue.

The following two tests fail because of a bad or invalid BIC:

- `mt103.3.syntOK.bicNoLongerValid.mbtest`

  The `mt103.3.syntOK.bicNoLongerValid.mbtest` test has a syntactically valid SWIFT message but a BIC that is no longer valid. Running this test should trigger the creation of a message on the SCENARIO2_BIC_REJECT queue.

- `mt103.4.syntOK.bicNotConnected.mbtest`

  The `mt103.4.syntOK.bicNotConnected.mbtest` test is just like the previous test. However, in this case, the problem is that the BIC is not connected to the SWIFT network. It should trigger the creation of a message on the SCENARIO2_BIC_REJECT queue.

The next two tests have a correct syntax and BIC but do not create a trace file because the amount is too low:

- `mt103.5.syntOK.bicOK.amtLOW.currEUR.mbtest`

  The `mt103.5.syntOK.bicOK.amtLOW.currEUR.mbtest` test is a test case with a valid syntax and BIC, where the amount is expressed in euro with an amount lower than €10,000. The expected result is a SWIFT message on the SCENARIO2_SWIFT_GATEWAY queue, and no trace file.

- `mt103.6.syntOK.bicOK.amtLOW.currUSD.mbtest`

  The `mt103.6.syntOK.bicOK.amtLOW.currUSD.mbtest` test is similar to the previous test, but the amount is expressed in U.S. dollars. The expected result is a message on the SCENARIO2_SWIFT_GATEWAY queue and no trace file.

The following three tests all have a correct syntax and BIC and an amount higher than €10,000. Therefore, they create a SWIFT message on the SCENARIO2_SWIFT_GATEWAY queue and a trace file on the file system.

- `mt103.7.syntOK.bicOK.amtHIGH.currUSD.mbtest`
- `mt103.8.syntOK.bicOK.amtHIGH.currEUR.mbtest`
- `mt103.9.syntOK.bicOK.amtHIGH.currGBP.mbtest`

Running the first test

To run the first test case:

1. Double-click the flow test file.
2. In the Unit Test Client (Figure 8-169), click **Send Message** to launch the test.

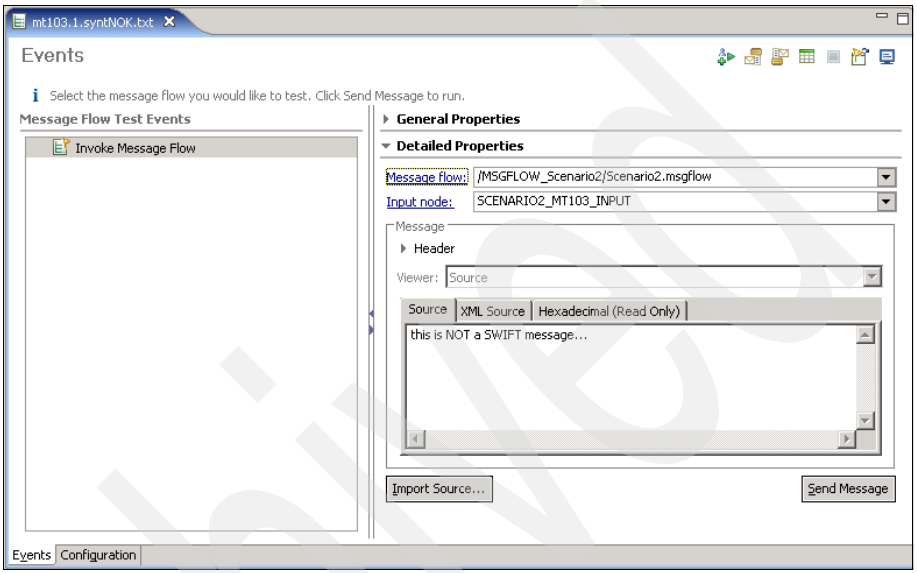

Figure 8-169 The Unit Test Client with the first test

3. If you see the Deployment Location window (Figure 8-170), point to the default execution group and click **Finish**.

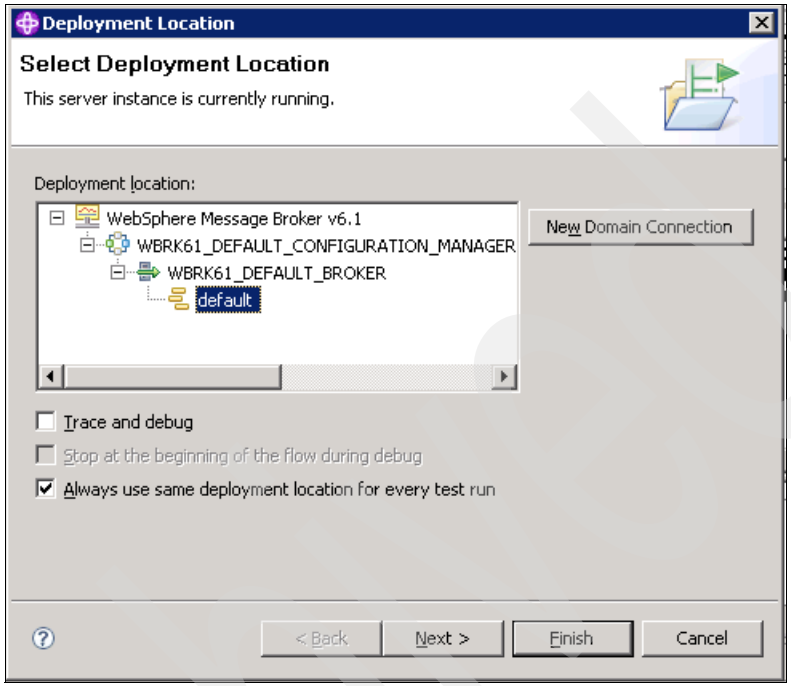

Figure 8-170 Selecting the deployment location

Figure 8-171 shows the result. Notice the message that a syntax error has occurred.

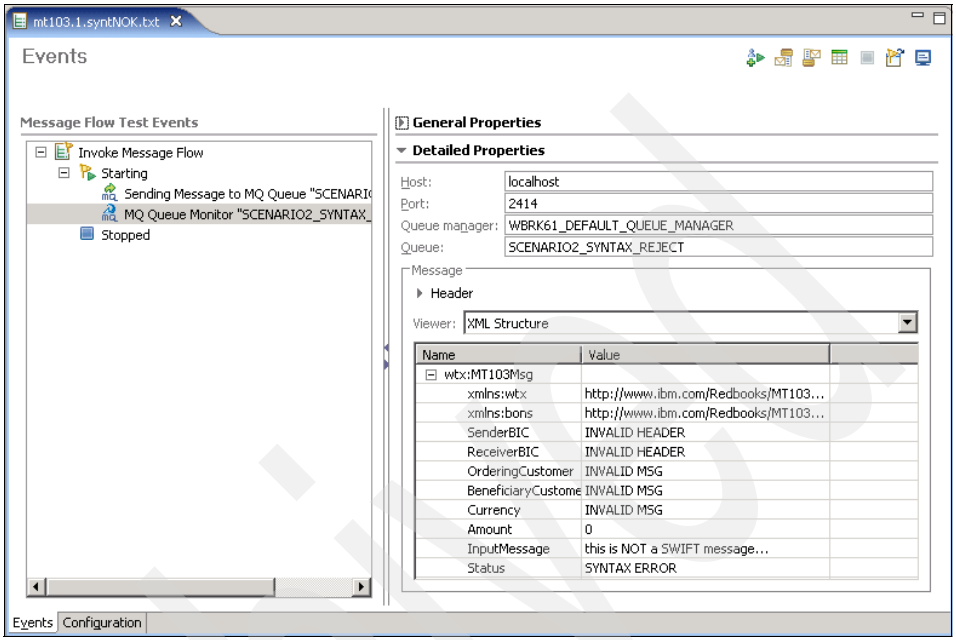

Figure 8-171 Result of test 1

Note: The way to execute the test cases is the same for all subsequent tests. Therefore, we do not show the window captures for all the tests here. However, you can find all window captures of all tests in Appendix B, “Running the test cases for the WebSphere Message Broker scenario” on page 713.

9

Integration scenario: WebSphere Transformation Extender on System z

In this chapter, we describe and build a scenario that uses WebSphere Transformation Extender on System z.

9.1 Scenario introduction

Company A has an order processing system that is handled by two COBOL programs, PROG1 and PROG2, that are called in sequence in a job control language (JCL). PROG1 receives the orders, checks if they are valid, and splits them. PROG1 writes the valid order records in a fixed length file that is read by PROG2, which processes the orders, as shown in Figure 9-1.

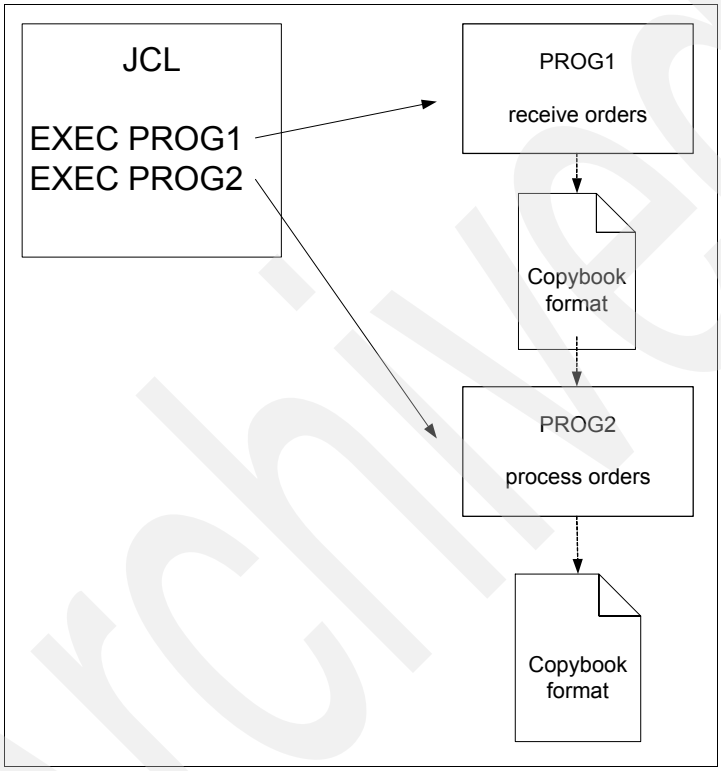

Figure 9-1 Original architecture

Company A acquired Company B that has its own order processing system. We want to integrate these systems while changing the existing systems as little as possible. The best scenario is not to change the existing systems at all.

Company B's order processing system has two parts:

- A COBOL program, PROG3, which is processing the orders coming from old partners
- A Java 2 Platform, Enterprise Edition (J2EE) application that processes the orders coming from new partners

The J2EE application reads XML files as input, while the COBOL program expects fixed length files.

An alternative is to change PROG1, which receives all orders, to dispatch to the other three systems, based on the customer ID, as shown in Figure 9-2. In this scenario, PROG1 also needs to convert the data to the format that is expected by each of the three systems. Unfortunately this will impact several other systems and will have a long development cycle. Also, producing XML from a COBOL program is not a trivial task and requires specific skills.

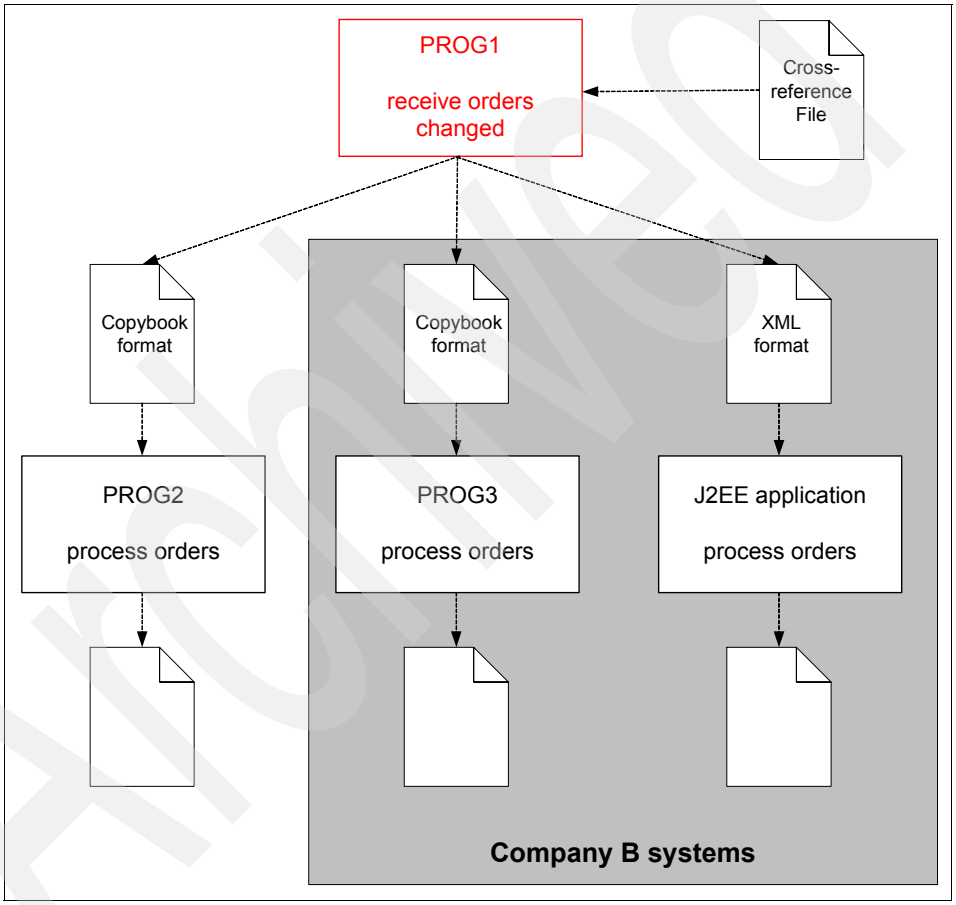

Figure 9-2 Alternative architecture without WebSphere Transformation Extender

We can solve this problem easily by using WebSphere Transformation Extender, without changing any of the existing applications. In the proposed new architecture, we have WebSphere Transformation Extender reading the fixed length file produced by PROG1. Based on the customer ID, WebSphere

Transformation Extender transforms the format of the data and dispatches it to the appropriate system.

Additionally, with WebSphere Transformation Extender, we can introduce a powerful logging and error handling mechanism. Figure 9-3 shows this proposed architecture.

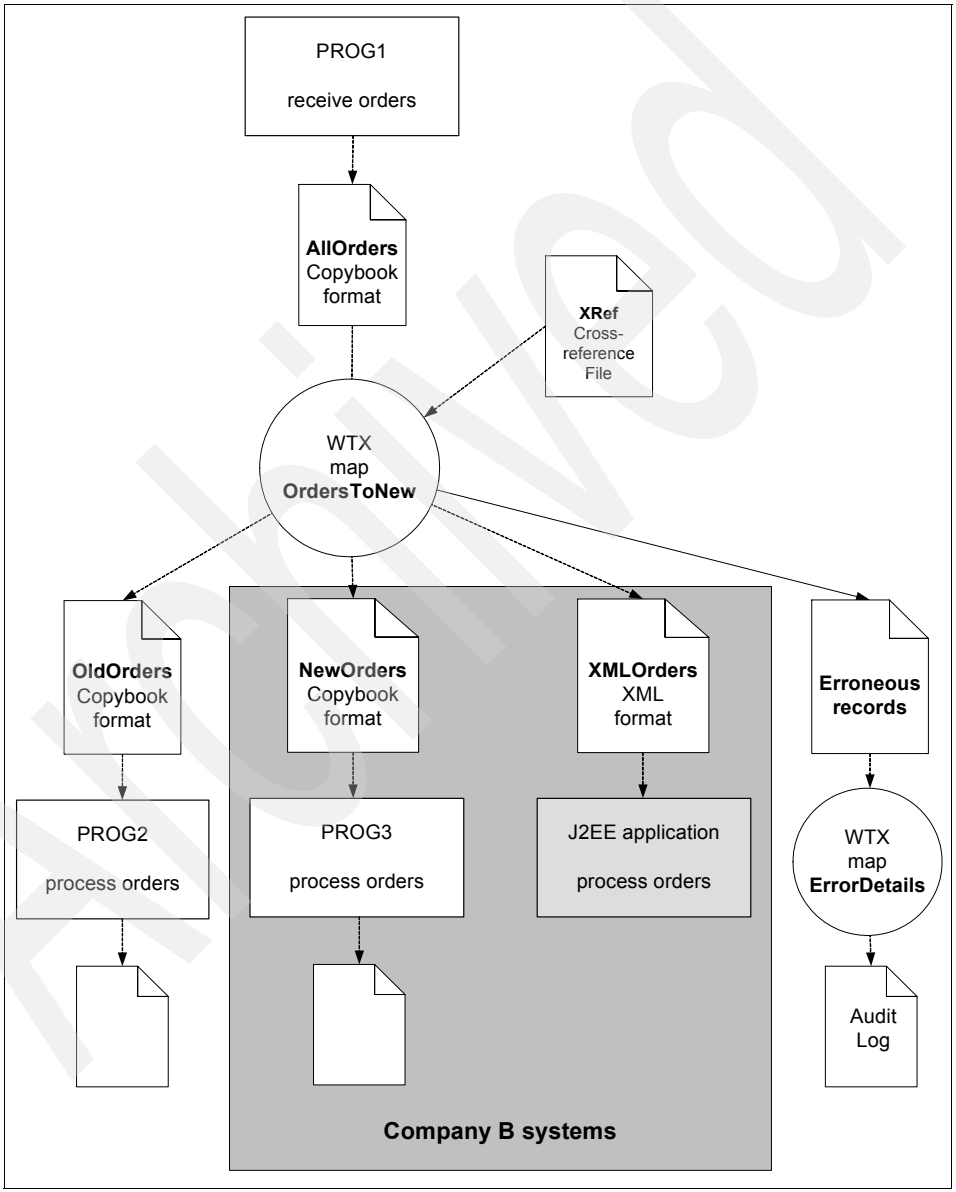

Figure 9-3 Proposed architecture with WebSphere Transformation Extender

9.2 Building the scenario

Besides the maps, four files are required to run this scenario:

AllOrders.txt The input text file that contains order records in fixed format. It is created by a COBOL program (PROG1).

Xref.txt The cross-reference input text file that is a semi-colon delimited data file.

Orders.cpy The COBOL copybook that contains the description of the order records for PROG2.

NewOrders.cpy The COBOL copybook that contains the description of the order records for PROG3.

Figure 9-4 shows tasks that are necessary to create the scenario.

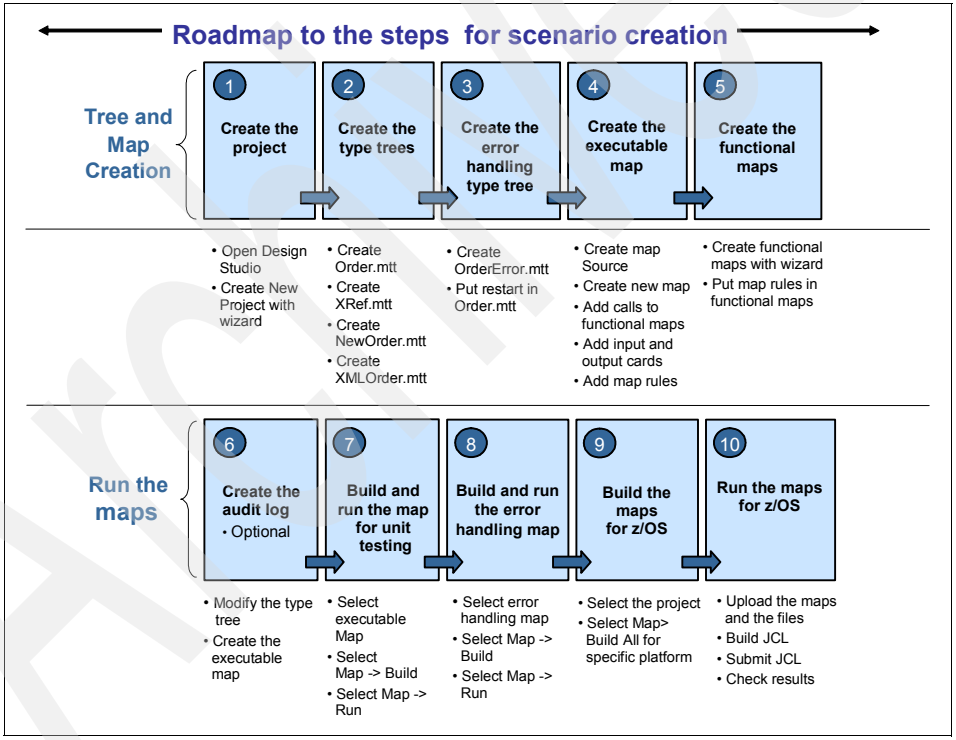

Figure 9-4 Roadmap for creating the System z scenario

The following tasks are required:

1. Create the project to contain all your artifacts.
2. Create the type trees used for input and output.
3. Create the error handling type tree.
4. Create the executable map.
5. Create the functional maps.
6. Optional: Create the audit log type tree and executable map.
7. Build and run the map for unit testing.
8. Build and run the error handling map.
9. Build the maps for z/OS.
10. Run the maps on z/OS.

Project Interchange file: The `WTX_Scenario3.zip` Project Interchange file, which is available for download, contains the complete working scenario. See Appendix E, “Additional material” on page 749, for details about this file. After you import this file, you can choose either of the following options:

- Only run the scenario described in this chapter, and study the type trees and maps to understand specific points.
- Create all the artifacts yourself, as explained in the steps before running the scenario.

To import the downloaded Project Interchange file into your Design Studio:

1. Open the Design Studio. Select **Start** → **Programs** → **IBM WebSphere Transformation Extender 8.2** → **Design Studio** → **Design Studio**.
2. Import the Project Interchange file:
   a. Select **File** → **Import**.
   b. In the Import window, select the source. Select **Other** → **Project Interchange**. Click **Next**.
   c. Browse to the directory where you have **WTX_Scenario3.zip** and select it.
   d. Select the **WTX_Scenario3** project and click **Finish**.

The WTX_Scenario3 project is created with all the artifacts that are necessary to run it in Design Studio. To run the project in a z/OS environment, check the readme file for this scenario and the steps in 9.2.10, “Running the maps on z/OS” on page 676.

If you want to follow the steps and create the scenario from the beginning, you can create a new project and copy the required files to it. These files are in the Misc folder of the imported Extender Project.

9.2.1 Creating the project

To create the project:

1. Open the Design Studio. Select **Start** → **Programs** → **IBM WebSphere Transformation Extender 8.2** → **Design Studio** → **Design Studio**.
2. Create a new project by selecting **File** → **New** → **Project**.
3. In the New Project wizard, select **Transformation Extender** → **Extender Project**. Click **Next**.
4. In the next window, enter the project name `WTX_Scenario3`, and click **Finish**.

9.2.2 Creating the type trees

The following type trees are used in this scenario:

Order.mtt Defines the `AllOrders.txt` file. This file is created by a COBOL program (PROG1). Therefore, we create the type tree with the COBOL copybook importer.

Xref.mtt Defines the cross-reference file. We do not have any definition of this file, apart of the file itself. Therefore, we create the type tree with the Text File importer.

NewOrder.mtt Describes the format of the orders that are handled by a COBOL program (PROG3). Therefore, we create the type tree with the COBOL copybook importer.

XMLOrder.mtt Describes the format of the orders that are sent to the J2EE application. We have the XML Schema Definition (XSD, schema) file. Therefore, we create the type tree with the XML schema importer.

OrderError.mtt Describes the file that contains the order records in error. We create this type tree manually, since there is no definition of this file.

Creating the Order.mtt type tree

The Order.mtt type tree is created by using the COBOL copybook importer. Figure 9-5 shows the COBOL copybook file that we used.

```
01                ENR-ORDERSIN.
             05           ORDER-NUMBER          PIC  X(10).
             05           CUSTOMER-ID           PIC  X(09).
             05           SHIP-TO-CODE          PIC  X(03).
             05           CONTACT-NAME          PIC  X(25).
             05           ORDER-DATE            PIC  X(08).
             05           WANT-DATE             PIC  X(08).
             05           CATALOG-NUMBER        PIC  X(12).
             05           QUANTITY-ORDERED      PIC  9(06).
             05           UNIT-OF-MEASURE       PIC  X(02).
                    88    UNIT-EACH             VALUE 'EA'.
                    88    UNIT-BOX              VALUE 'BX'.
                    88    UNIT-CASE             VALUE 'CS'.
                    88    UNIT-PIECE            VALUE 'PC'.
             05           UNIT-PRICE            PIC  9(6).99.
             05           MESSAGE1              PIC  X(25).
             05           MESSAGE2              PIC  X(25).
```

Figure 9-5 COBOL copybook for orders

To create the `Order.mtt` type tree:

1. Create the type tree by using the COBOL copybook importer.
2. Add the validation rules in the `Order.mtt` type tree.
3. Analyze and save the type tree.

Creating the type tree by using the COBOL copybook importer

In our scenario, we imported the `Orders.cpy` and `NewOrders.cpy` copybooks into the project so that we have all artifacts together. To import these files in the project:

1. Right-click **WTX_Scenario3** and select **Import** → **File System**.
2. Browse to the directory where you have your files and select the files that you want to import.
3. In the next window, verify that the Into folder: field contains `WTX_Scenario3`. Click **Finish**.

The files are now in the Misc folder, under the project. You can also work with the copybooks in any other directory on your computer, if you choose.

Note: You browse through the files in your workspace the same way you do with other directories. Check the path of your workspace directory, in which you will find all your projects.

To create the `Order.mtt` type tree:

1. Expand the **WTX_Scenario3** project, right-click the **Type Trees** folder and select **Import** → **COBOL copybook**.
2. In the COBOL copybook Importer wizard, browse to the folder where the `Orders.cpy` copybook is located. Click **Next**.
3. In the Select byte and character sets window, accept the defaults (**NATIVE**), and click **Next**.
4. In the next window, for the parent folder, select **WTX_Scenario3**. For File name, type `Order.mtt`. Then click **Next**.
5. Verify that there are no errors in the creation of the type tree. Click **Finish**.
6. Open the `Order.mtt` type tree and complete it:

   A COBOL copybook is a description of a record. We want to use the type tree to validate a file with several records. Therefore, we must put it in the type tree definition.

   a. Add a new element to the type tree. Define it as a Group that contains several records (ENR-ORDERSIN), as shown in Figure 9-6.

      i. Insert a new element, change the Class to **Group**, and name the new group `File`.

      ii. Drag **ENR-ORDERSIN** to the **File** component and set the range (Min `0`, Max `s`).

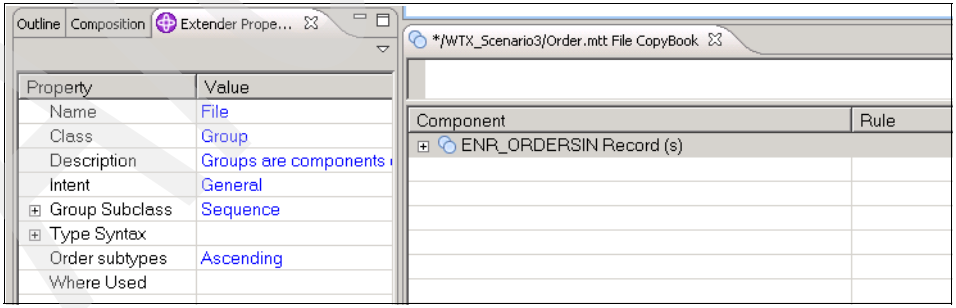

Figure 9-6 File group - Properties view

   b. Check that the decimal values are correctly defined in the type tree. In our case we have the UNIT-PRICE defined in the copybook as `PIC 9(6).99`.

We want to make sure that we always have two decimals after the decimal point ("."). Open the Properties view for the UNIT-PRICE item and navigate to Item Subclass of **Places**. Set Decimal Min and Decimal Max to 2, as shown in Figure 9-7.

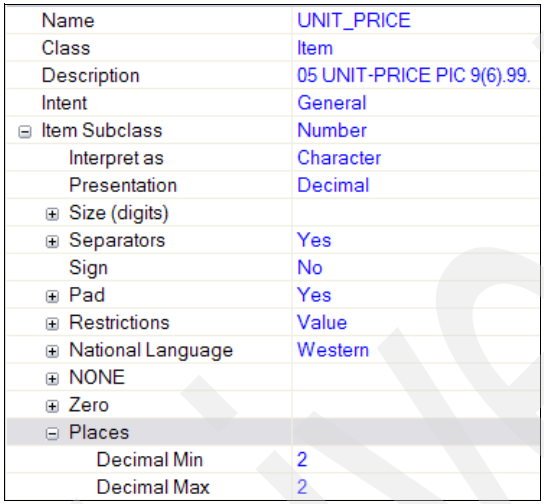

Figure 9-7 UNIT-PRICE item - Properties view

c. Include a record separator (new line) at the end of each record in the type tree definition for clarity. Open the Properties view for the **ENR_ORDERSIN** record element. Expand the Type Syntax field and change Terminator to `Literal` and for Value, enter `<NL>` (Figure 9-8).

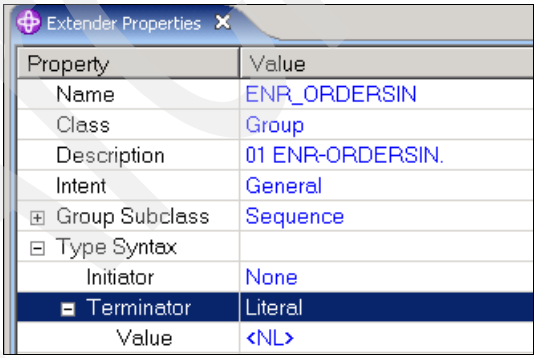

Figure 9-8 ENR_ORDERSIN item - Properties view

7. Save the type tree.

Note: In our scenario, we used an input file where each record is terminated by an <NL> character to make it easier to understand. In a pure z/OS COBOL copybook, the records follow each other without any separation. Therefore, this step is unnecessary.

Adding the validation rules to the Order.mtt type tree

For the `Order.mtt` type tree, we have two rules:

- The ship-to-code field must be numeric.
- The order date must precede the want date.

To add these rules in the type tree:

1. Open the group that represents the **ENR_ORDERSIN** record by double-clicking it.
2. In the SHIP_TO_CODE field, type `ISNUMBER($)`, and then press Enter.
3. In the WANT_DATE field, type `WHEN(PRESENT($), $ > ORDER_DATE Field)`.

Figure 9-9 shows the rules in the record group.

Tip: To drag ORDER_DATE to the WANT_DATE rule, hold down the SHIFT key, select **ORDER_DATE**, and then drag it to the correct place in the rule.

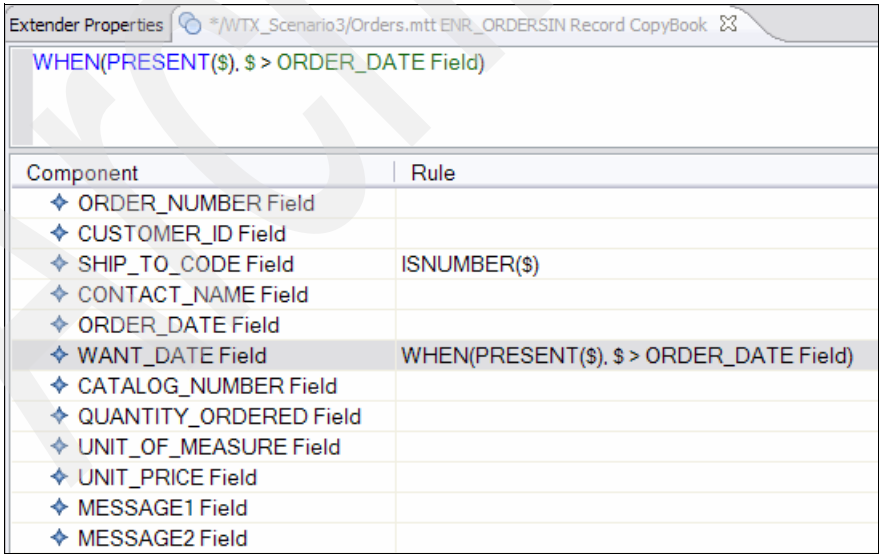

Figure 9-9 Validation rules in the Order.mtt type tree

Analyzing and saving the Order.mtt type tree

In the menu bar, select **Tree** → **Analyze** → **Structure and Logic**. Check that there are no errors. Save the type tree. Figure 9-10 shows the complete type tree.

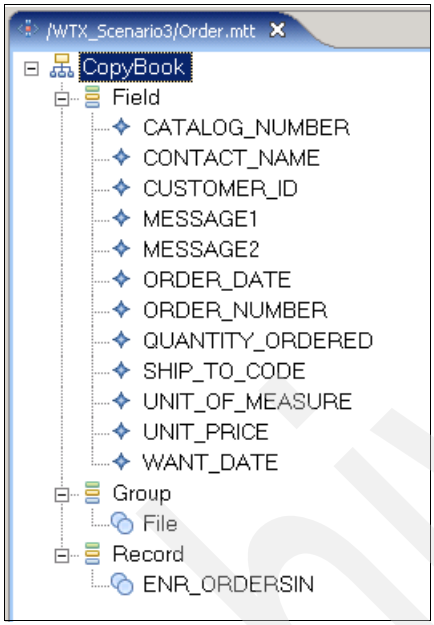

Figure 9-10 Complete Order.mtt type tree

Tips: When creating a type tree from a COBOL copybook, keep in mind the following tips:

- Create a file group and define it as a group of records.
- Check if the decimal values are correctly defined in the type tree.

Creating the Xref.mtt type tree

The `Xref.mtt` type tree is a simple one, based on a semi-colon separated file, as shown in Figure 9-11.

```
ABCWIDGET;A123;ABC Widgets
MARKSBROS;M621;Marks Bros Ltd.
TSTPRINCE;T231;T.S.T. Prince Industries
ZZZZXXXX;U525;Unknown Customer
```

Figure 9-11 Xref file

To create the `Xref.mtt` type tree:

1. Create a type tree by using the Text File Importer.
2. Open the type tree to complete it.
3. Analyze and save the type tree.

Creating the type tree by using the Text File Importer

Since we have the cross-reference file, we can create the `Xref.mtt` type tree by using the Text File Importer:

1. Expand the **WTX_Scenario3** project, right-click the **TypeTrees** folder, and select **Import** → **Text File**.
2. In the Text File Importer wizard, browse to where the `Xref.txt` file is located. Click **Next**.
3. In the Select byte and character sets window, accept the defaults and click **Next**.
4. In the next window, select **WTX_Scenario3** as the parent folder. For File name, enter `Xref.mtt`. Click **Next**.
5. In the Text File Importer window (Figure 9-12), create the type tree:

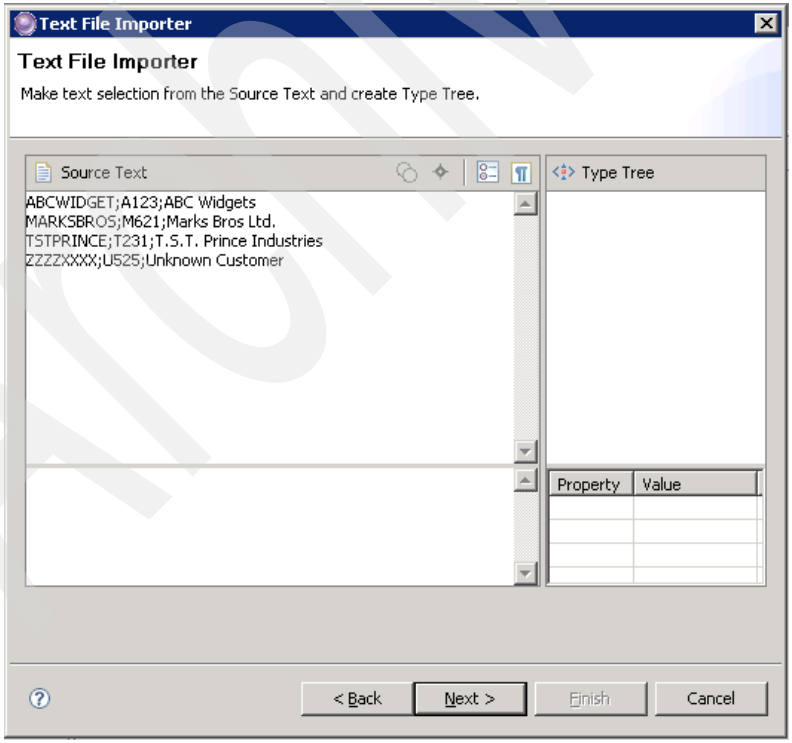

Figure 9-12 Text File Importer window

a. Select the whole text:

```
ABCWIDGET;A123;ABC Widgets
MARKSBROS;M621;Marks Bros Ltd.
TSTPRINCE;T231;T.S.T. Prince Industries
ZZZZXXXX;U525;Unknown Customer
```

b. Click the **Group** button ( ) to add a group to the type tree.

c. In the Properties view, in the bottom right corner of the window, change the name of the group to `File`.

d. Select the first line, which is `ABCWIDGET;A123;ABC Widgets`.

e. Click the **Group** button ( ) to add a group to the type tree.

f. In the Properties view, change the name of the group to `Record`.

g. Select the first element of the first line, which is `ABCWIDGET`.

h. Click the **Item** button ( ) to add an item to the group.

i. In the Properties view, change the name of the item to `OldCustomerID`.

j. Select the second element of the first line, which is `A123`.

k. Click the **Item** button ( ) to add an item to the group.

l. In the Properties view, change the name of the item to `NewCustomerID`.

m. Select the third element of the first line, which is `ABC Widgets`.

n. Click the **Item** button ( ) to add an item to the group.

o. In the Properties view, change the name of the item to `CustomerName`.

Figure 9-13 on page 637 shows the completed type tree in the Text File Importer window.

p. Click **Next**, and the type tree is generated.

Tip: Before you attempt to change the name of the element in the Properties window, make sure that you have the right element of the type tree selected. You might have to press Enter to force the change and see the new name in the Type Tree window. Note that the hierarchy of the Text File importer builds in the Type Tree window and that you might have to drill down to get to the desired element.

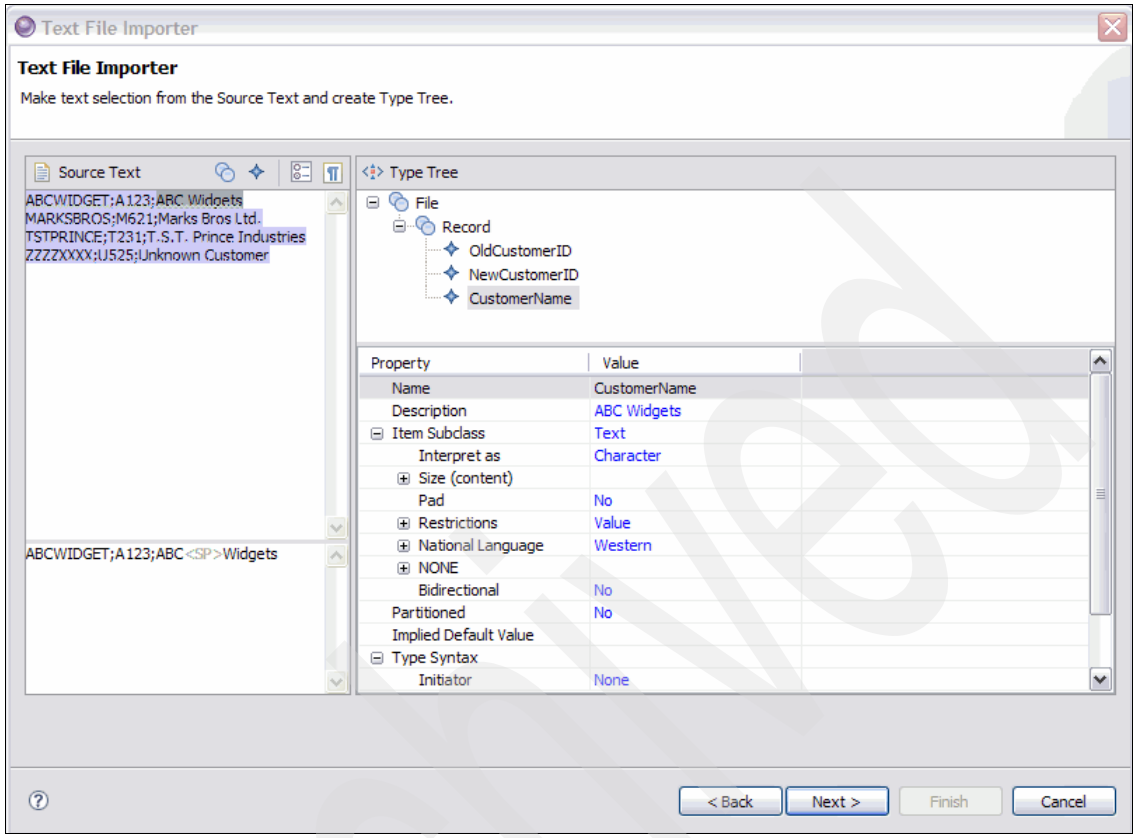

Figure 9-13 Xref.mtt type tree in Text File Importer window

6. Check that there are no errors in the creation of the type tree. Click **Finish**.

Element name change in the type tree: It is possible to change the name of the elements in the type tree as soon as they are created. However, sometimes the name looks as though it is changed, but it is not. To ensure that the names are changed, change them in the Properties view, in the bottom right corner of the Text File Importer window as we did in this task.

Opening and completing the Xref.mtt type tree

Open the type tree and complete it:

1. For Type Syntax, check the Initiator and Terminator and change them if necessary.
2. For Group Subclass, check the Format and Delimiter and change them if necessary.

3. For items defined as Number, check the format and change it if necessary.
4. Check the range of the fields in each group and change them if necessary.
5. Check the size of the contents in the definition of the items and change it if necessary.
6. Check the description of each element in the type tree. The Text File Importer wizard copies the contents of the fields that are selected as the description. You might want to add a better description.

In our example, the type tree is simple. All the item elements are of the Text subclass, with no specific syntax. Therefore, all default values are acceptable, with one exception. You must change the CustomerName field, by selecting **Item Subclass** → **Size** → **Max** → **Characters** and specifying a value of 30. Also, change the descriptions of the fields.

For the group elements, make the following changes:

1. For the Record element, define the Group Subclass, Format, and Component Syntax.
2. For Type Syntax, define the Terminator value.

Figure 9-14 shows the Properties view for the Record group.

Property	Value
Name	Record
Class	Group
Description	
Intent	General
⊟ Group Subclass	Sequence
Partitioned	No
⊟ Format	Explicit
Track	Content
⊟ Component Syntax	Delimited
⊟ Delimiter	Literal
Value	;
Ignore case	No
⊞ National Language	Western
Location	Infix
⊟ Type Syntax	
Initiator	None
⊟ Terminator	Literal
Value	<NL>
Ignore case	No
⊞ National Language	Western
Release	None
Empty	None
Order subtypes	Ascending
⊞ Where Used	

Figure 9-14 Record group properties

The File group is a Sequence group with Implicit format and no syntax (no initiators or terminators). Figure 9-15 shows the Properties view for the File group.

Property	Value
Name	File
Class	Group
Description	
Intent	General
⊟ Group Subclass	Sequence
Partitioned	No
⊟ Format	Implicit
Floating Component	
Component Syntax	None
⊟ Type Syntax	
Initiator	None
Terminator	None
Release	None
Empty	None
Order subtypes	Ascending

Figure 9-15 File group properties

For the File group, make sure that you have the correct range of the Record component (s), as shown in Figure 9-16.

Component
⊟ Record Group (s)
OldCustomerID Field
NewCustomerID Field
CustomerName Field

Figure 9-16 File group component view

Analyzing and saving the Xref.mtt type tree

In the menu bar, select **Tree** → **Analyze** → **Structure and Logic**. Ensure that there are no errors. Save the type tree. Figure 9-17 shows the completed type tree.

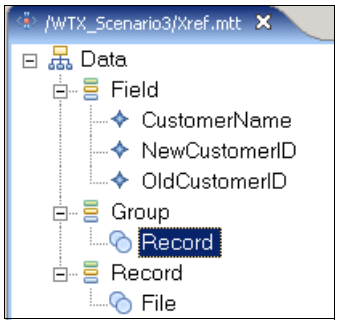

Figure 9-17 Completed Xref.mtt type tree

Creating the NewOrder.mtt type tree

To create the NewOrder type tree:

1. Create a type tree by using the COBOL copybook importer. The steps are the same for creating the `Order.mtt` type tree as explained in “Creating the Order.mtt type tree” on page 630. Import the **NewOrders.cpy** copybook. Remember to add a File group made of some (s) Record elements.
2. Analyze the type tree:
   a. In the menu bar, select **Tree** → **Analyze** → **Structure and Logic**.
   b. Check that there are no errors.
   c. Save the type tree.

Figure 9-18 shows the completed type tree.

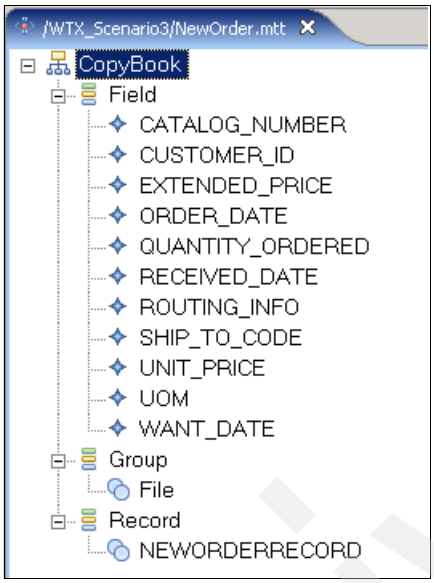

Figure 9-18 Complete NewOrder.mtt type tree

Creating the XMLOrder.mtt type tree

Usually you get the XML schema (XSD or document type definition (DTD)) and use the appropriate importer to create the type tree. For this scenario, we did not have a schema. Therefore, we must create one. For simplicity, we assume that the XML contains the same fields as the ones in the NewOrders COBOL copybook.

Create the XSD file based on the `NewOrder.mtt` type tree:

1. Open the **NewOrder.mtt** type tree.
2. Highlight the **File** group and from the menu bar, select **Tree** → **Export As Schema** (Figure 9-19 on page 642).

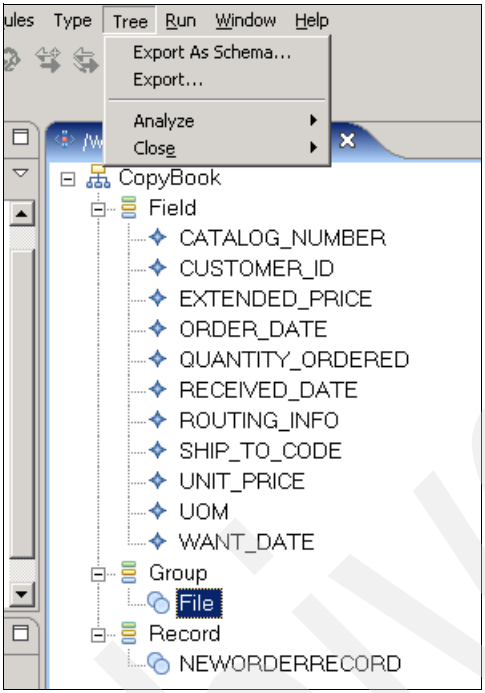

Figure 9-19 Exporting a type tree as a schema

3. In the window that opens, accept the defaults for the names for the input, output, and map files. Click **OK**.

   An XSD schema and a map are generated. The map has the original type tree in the input card and the XSD schema as the output card. For this scenario we do not use the map generated automatically, only the XSD.

4. Delete the `NewOrder.mms` map that is created in the Map Files folder.

5. Close the schema file.

Note: You can create an XML schema from any type tree. Select the element that you want to start your schema, which is usually the definition of the file. Then from the menu bar, select **Tree** → **Export As Schema**.

There are two possibilities when creating a map that reads or generates XML files:

- The classical one, which uses a type tree generated from an XML schema
- The new one, which uses the XSD file directly at the map

  You do not need to generate a type tree, but rather use the XSD file directly as an input or output card.

In this scenario, we create the type tree from the XSD file to show the functionality of the XML Schema Importer. Otherwise we can use the XSD file directly in the map, because it is a type tree, as shown in Figure 9-20.

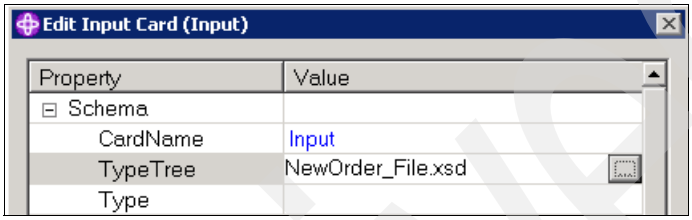

Figure 9-20 Using an XSD file as an input card

To create the XMLOrder type tree:

1. Create a type tree by using the XML Schema Importer.
2. Analyze and save the type tree.

Creating a type tree by using the XML Schema Importer

To create a type tree by using the XML Schema Importer:

1. Expand the **WTX_Scenario3** project, right-click the **Type Trees** folder and select **Import** → **XML Schema**.
2. In the XML Schema Importer wizard, browse to the folder where the XSD file is located. Click **Next**.
3. In the Specify data type window, accept the defaults.
4. Select **WTX_Scenario3** as the parent folder. For File name, type `XMLOrder.mtt`. Click **Next**. The type tree is generated.
5. Check that there are no errors in the creation of the type tree. Click **Finish**.

Analyzing and saving the XMLOrder.mtt type tree

For the first time that we analyze the type tree, we get several warnings. This is because some unused items are deleted. If we analyze it again, the warnings disappear.

In the menu bar, select **Tree** → **Analyze** → **Structure and Logic**. Repeat this selection. Check that there are no errors. Save the type tree. Figure 9-21 shows the completed type tree.

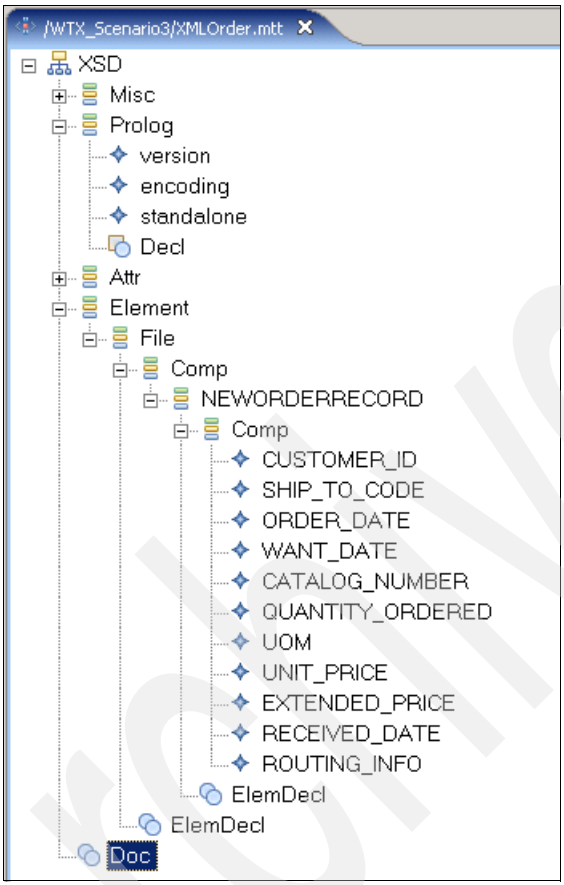

Figure 9-21 Completed XMLOrder.mtt type tree

9.2.3 Creating the error handling type tree

The OrderError.mtt type tree is used to format the file that contains the records that present any error, that is records that are not validated by the Order.mtt type tree. Figure 9-22 shows the output file that we defined.

```
Map version  1.00
Input File - C:\IBM\wtx\workspace\WTX_Scenario3\LegacyError.txt
Process Date:  080916
Time: 16:29:42

The following records are in error:

XRF-1A-150ABCWIDGET003
2008042820080510KL-MRN19-BLU000340ET000012.47May substitute
KL-MRN-001
LLM-8J-679ComputerS01BWhite, John
2008050120080502EV-RRC33-YLW000010BX000150.00No substitutions:Ship
Next-Day Air
BRA-8J-150101564250013Red, Jane
2008090120080502EV-RRC32-BLU000100BX000069.90Airmail
CEP-8J-552020505048013Brown, Jack
2008050120080502VE-RRC33-BLU000100MA000069.90No substitutions:Ship
Next-Day Air
```

Figure 9-22 ErrorDetails file

Since there are no existing artifacts that we can use to generate this type tree, we create the OrderError type tree by using the following steps:

1. Create a type tree manually by using the New Type Tree wizard.
2. Analyze and save the type tree.

Creating a type tree manually

To create the type tree manually:

1. Expand the **WTX_Scenario3** project, right-click the **Type Trees** folder, and select **New** → **Type Tree**.
2. In the New Type Tree wizard, select **WTX_Scenario3** as the parent folder. For File name, enter OrderError. Then click **Finish**.

3. In the empty type tree that opens, insert the elements:
   a. Under the Root type, add a Category type named `Field`.
   b. Add the Item types (simple objects) to the type tree under the Field category for the fields described in Table 9-1.

Table 9-1 Item Fields in the OrderError.mtt type tree

Field name	Type	Min	Max	Remarks
FileName	Text			Initiated by: “Input File:“ Terminated by a <NL> symbol
ProcessDate	Date			Format: YYMMDD Initiated by: “Process Date:” Terminated by a <NL> symbol
ProcessTime	Time			Format: HH24:MM:SS Initiated by: “Time:” Terminated by two <NL> symbols
Message	Text	0	50	Terminated by two <NL> symbols
ErrorText	Text			Text item of unlimited length Terminated by a <NL> symbol
MapVersion	Text			Initiated by “Map Version”<SP><SP>1.00” Terminated by a <NL> symbol

   c. Add a **Group** type to the type tree under the Root type named File. Populate the group as shown in Figure 9-23.

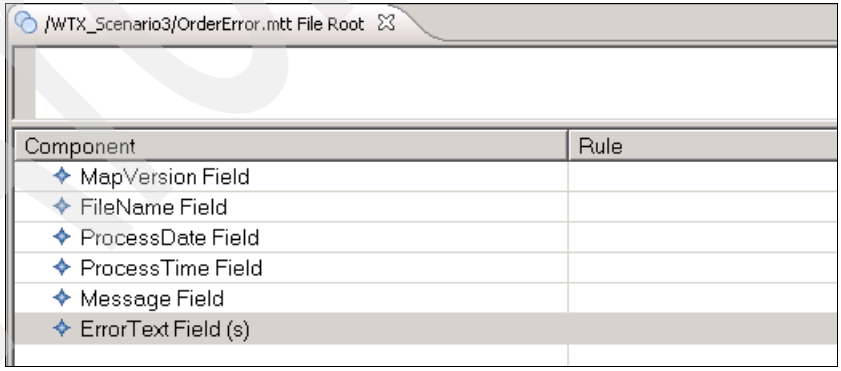

Figure 9-23 ErrorFile group

   d. Save the type tree.

Analyzing and saving the OrderError.mtt type tree

To analyze the `OrderError.mtt` type tree, in the menu bar, select **Tree** → **Analyze** → **Structure and Logic**. Check that there are no errors. Save the type tree. Figure 9-24 shows the completed type tree.

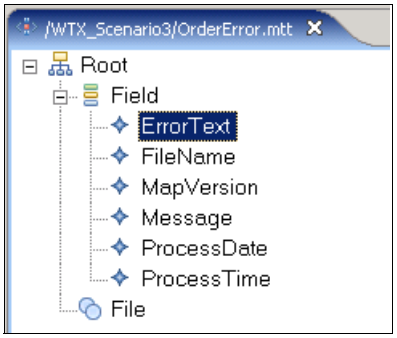

Figure 9-24 Complete OrderError.mtt type tree

The last step to implement the error handling is to put a *restart* function in the input record `Order.mtt` type tree. If we do not put in the restart function, the map stops processing when it encounters the first error and the error file is not produced.

Open the **Order.mtt** type tree, and then find and double-click the **File** element. In the File component window, right-click **ENR_ORDERSIN Record(s)** and select **Restart** from the menu.

Figure 9-25 shows the completed `Order.mtt` type tree with the restart function.

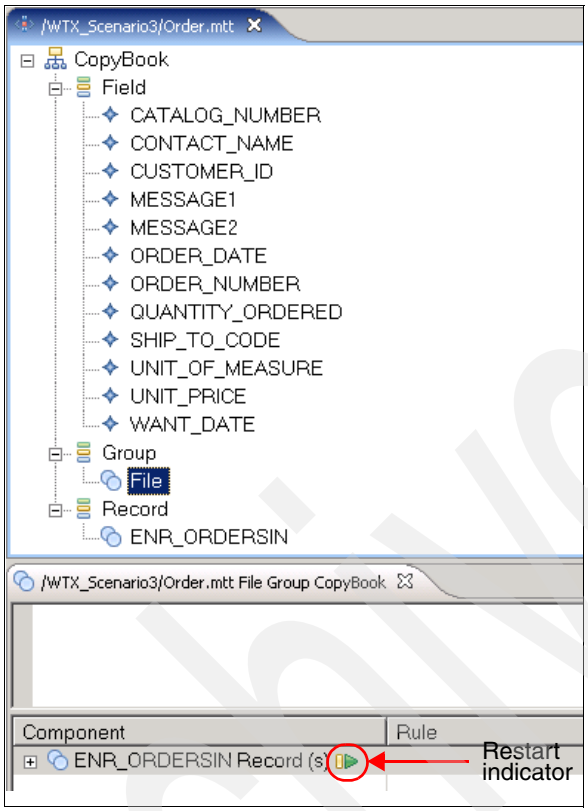

Figure 9-25 The Order.mtt type tree with the restart function

9.2.4 Creating the executable map

Our executable map, named OrdersToNew, contains two input cards:

- Input card #1 describes the file that contains the orders that are created by PROG1:
  - Type tree: `Order.mtt`
  - Input file: `AllOrders.txt`
- Input card #2 describes the cross-reference file:
  - Type tree: `Xref.mtt`
  - Input file: `Xref.txt`

The OrdersToNew executable map also contains four output cards:

- Output card #1 describes the file that is read by PROG2:
  - Type tree: `Order.mtt`
  - Output file: `OldOrders.txt`
- Output card #2 describes the file that is read by PROG3:
  - Type tree: `NewOrder.mtt`
  - Output file: `NewOrders.txt`
- Output card #3 describes the XML file that is read by the J2EE application:
  - Type tree: `XMLOrder.mtt`
  - Output file: `XMLOrders.xml`
- Output card #4 describes the file containing the records in error:
  - Type tree: `OrderError.mtt`
  - Output file: `OrderErrors.txt`

Each output card creates a file to be consumed by one of the systems in the new architecture: PROG2, PROG3, and the J2EE application. Each file contains only the order records for the customers associated with their respective systems.

In the appropriate map rule cell in the output cards, we place the call to the functional maps. The functional maps do the mapping of one existing record to one order record for one of the systems. Before we call each functional map, we check whether the customer ID corresponds to a user who is associated with the system (PROG2, PROG3, or XML). For this scenario, we decided to base our selection criteria on the customer ID:

- Company A customers have a numeric user ID.
- Company B customers have alphanumeric user IDs. Company B has two different systems: one for former customers and one for new customers. For former customers, we must convert the user ID based on a cross-reference table. For new customers, we use the customer ID as is.

After completing the functional map call in the executable map, we use the Functional Map Wizard to create the functional maps.

To create the executable map:

1. Expand the **WTX_Scenario3** project, right-click the **Map Files** folder and select **New** → **Map Source**.
2. In the Map Source Wizard, select **WTX_Scenario3** as the parent folder. For File name, enter `OrdersToNew`. Click **Finish**.
3. In the OrdersToNew map source that opens, click the **New Map** button ( ) to create an executable map. Change the map name to `OrdersToNew`.

4. Add input card #1 with the following settings (Figure 9-26):

   a. For TypeTree, type `Order.mtt`.
   b. For FilePath (input file), type `AllOrders.txt`.
   c. Click **OK**.

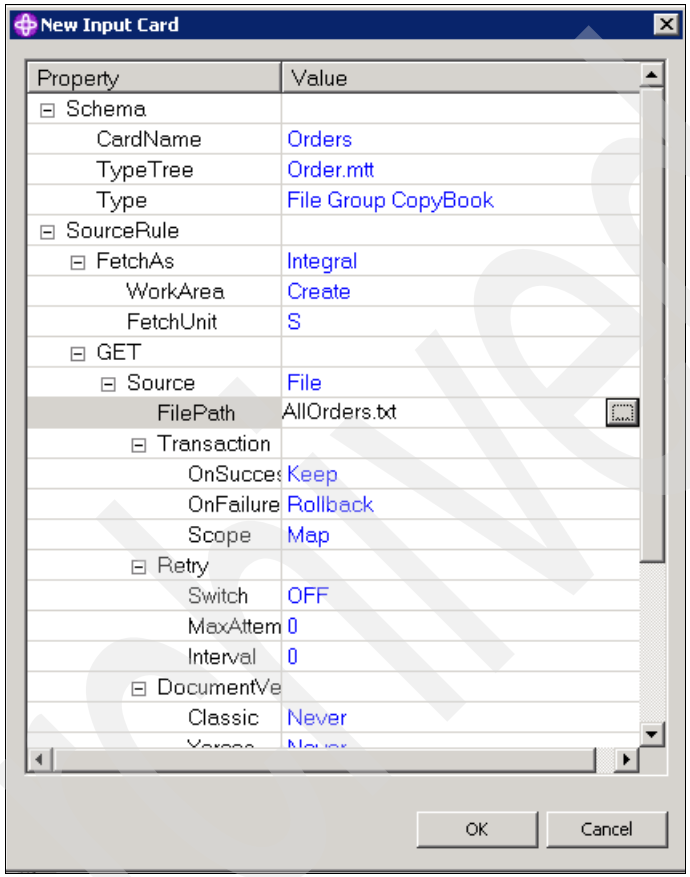

Figure 9-26 Input card #1 settings

5. Add input card #2 with the following settings (Figure 9-27):

   a. For TypeTree, type `Xref.mtt`.
   b. For FilePath (input file), type `Xref.txt`.
   c. Click **OK**.

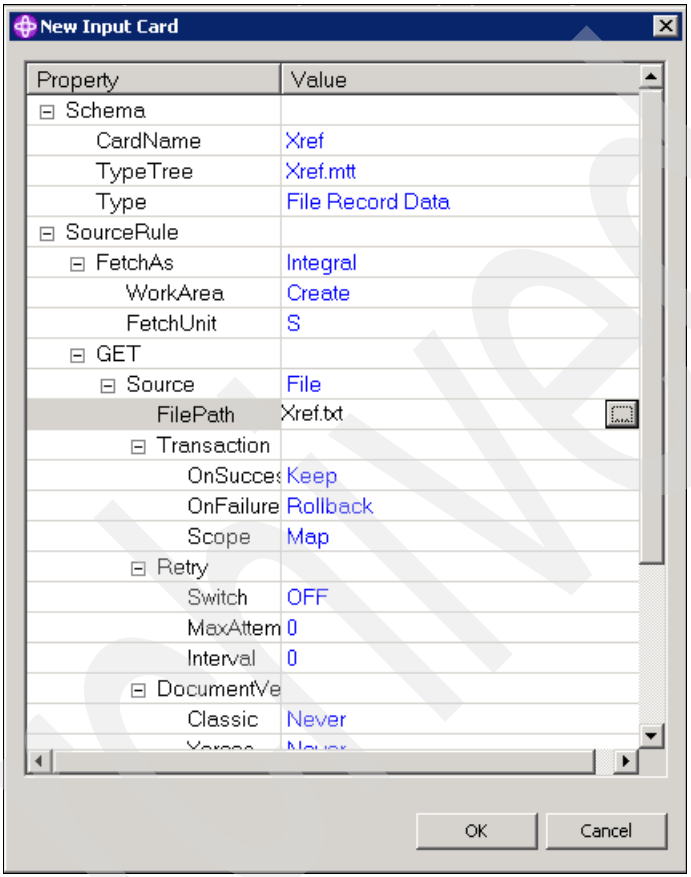

Figure 9-27 Input card #2 settings

6. Add output card #1 with the following settings (Figure 9-28):
   a. For TypeTree, type `Order.mtt`.
   b. For FilePath (output file), type `OldOrders.txt`.
   c. Click **OK**.

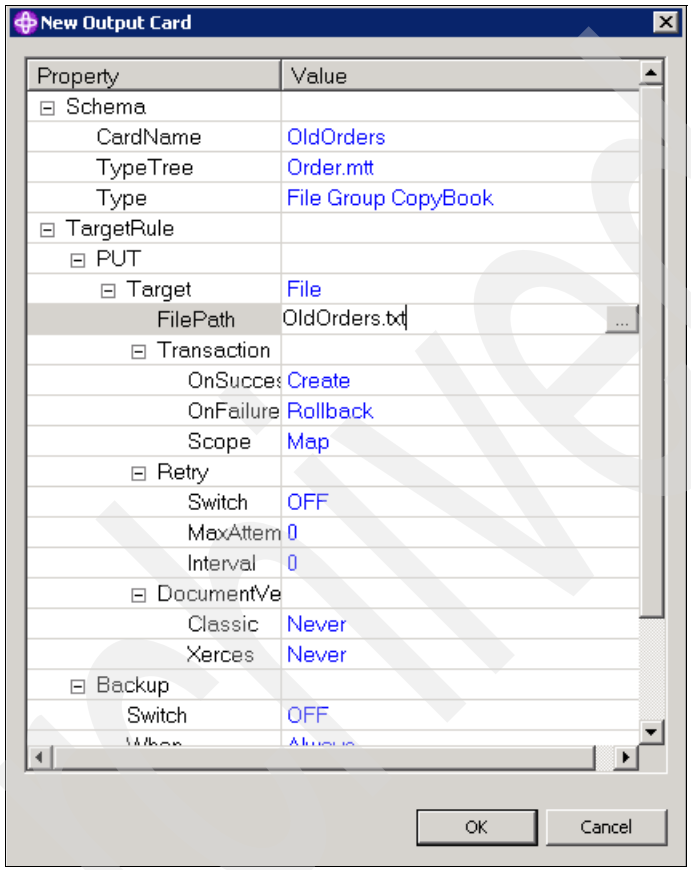

Figure 9-28 Output card #1 settings

7. Add the map rules to output card #1. The former system handles the customers from Company A that have a numeric customer ID. Since this is the existing system, we do not need to change anything in the records. The output card #1 (Figure 9-29) has the following rule:

```
=IF( ISNUMBER( LEFT( CUSTOMER_ID Field:ENR_ORDERSIN Record:Orders,1
) ), ENR_ORDERSIN Record:Orders )
```

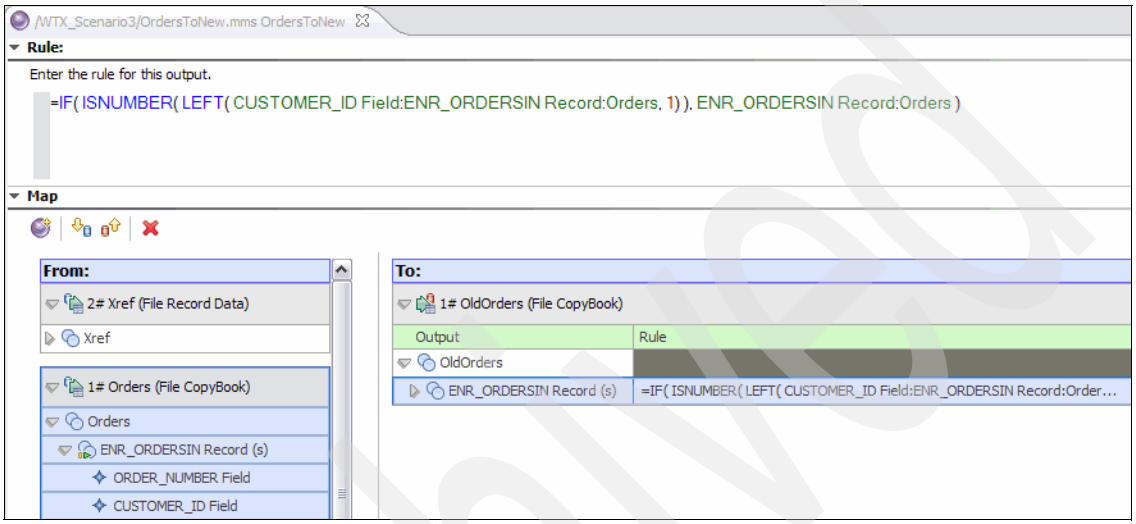

Figure 9-29 Output card #1 rules

8. Add output card #2 with the following settings (Figure 9-30):

   a. For TypeTree, type `NewOrder.mtt`.
   b. For FilePath (output file), type `NewOrders.txt`.
   c. Click **OK**.

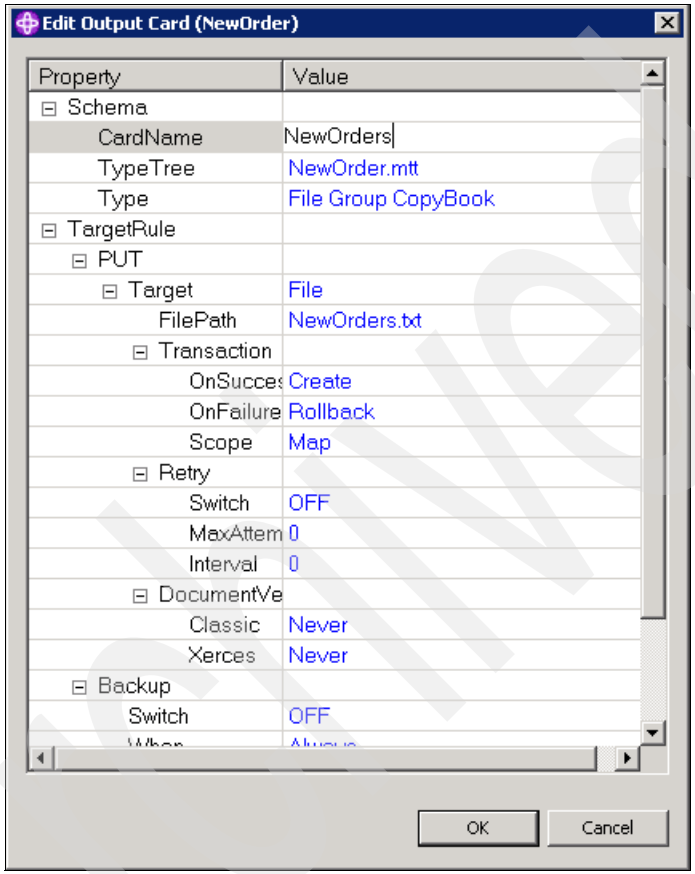

Figure 9-30 Output card #2 settings

9. Add the map rules to output card #2. This card generates the COBOL records for PROG3, from Company B. This system handles the former customers from Company B whose customer IDs must be converted to new customer IDs based on a cross-reference table. We need a functional map to transform each record. Output card #2 (Figure 9-31) uses the following rule:

```
=IF( NOT( ISNUMBER( LEFT( CUSTOMER_ID Field:ENR_ORDERSIN
Record:Orders,1 ) ) ),
      F_MakeOrderPROG3( ENR_ORDERSIN Record:Orders,
LOOKUP(NewCustomerID Field:Record Group:Xref, OldCustomerID
Field:Record Group:Xref =  CUSTOMER_ID Field:ENR_ORDERSIN
Record:Orders ) ) )
```

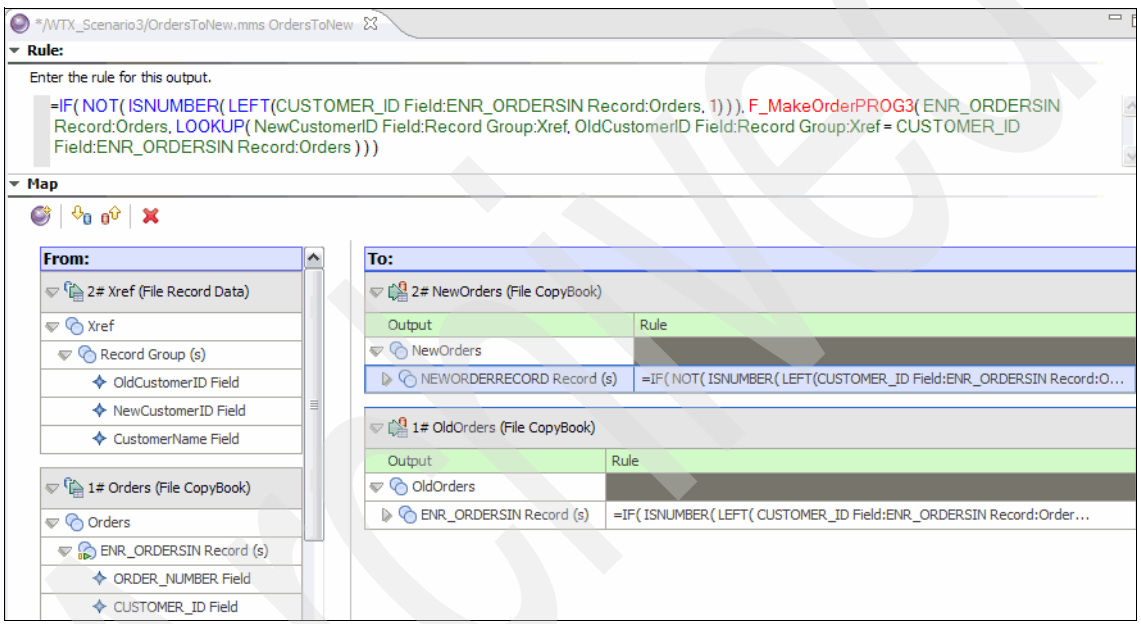

Figure 9-31 Output card #2 rules

10. Add output card #3 with the following settings (Figure 9-32):

    a. For TypeTree, type `XMLOrder.mtt`.
    b. For FilePath (output file), type `XMLOrders.xml`.
    c. Click **OK**.

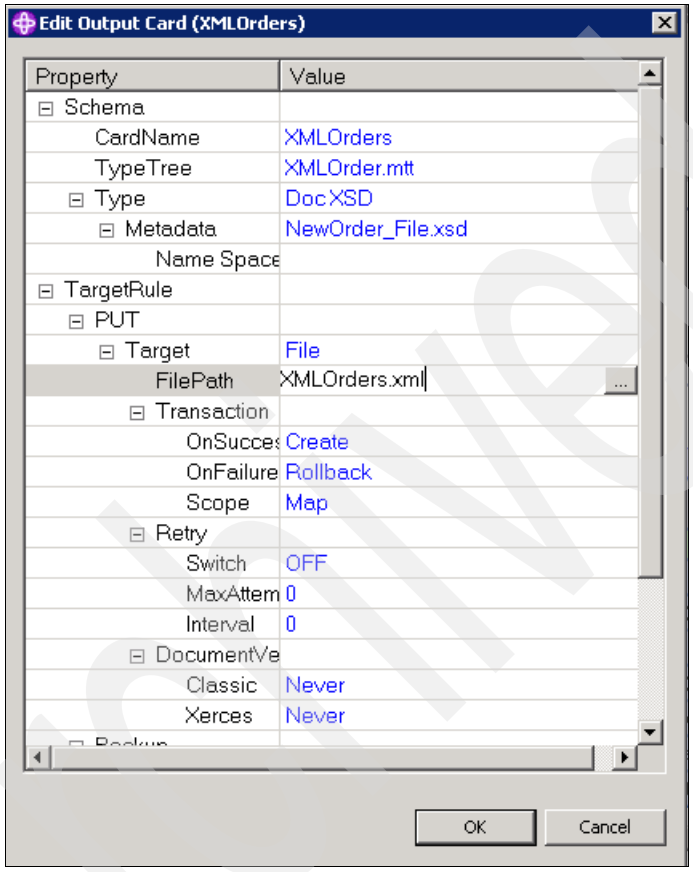

Figure 9-32 Output card #3 settings

11. Add the map rules to output card #3. This card generates XML records for the J2EE application from Company B. This system handles the new customers from Company B whose customer IDs do not need any conversion. We need a functional map to transform each record. Output card #3 (Figure 9-33) uses the following rule:

```
=IF( NOT( ISNUMBER( LEFT( CUSTOMER_ID Field:ENR_ORDERSIN
Record:Orders, 1 ) ) ) &
         NOT( MEMBER( CUSTOMER_ID Field:ENR_ORDERSIN Record:Orders,
OldCustomerID Field:Record Group:Xref ) ),
                F_MakeOrderXML(ENR_ORDERSIN Record:Orders ))
```

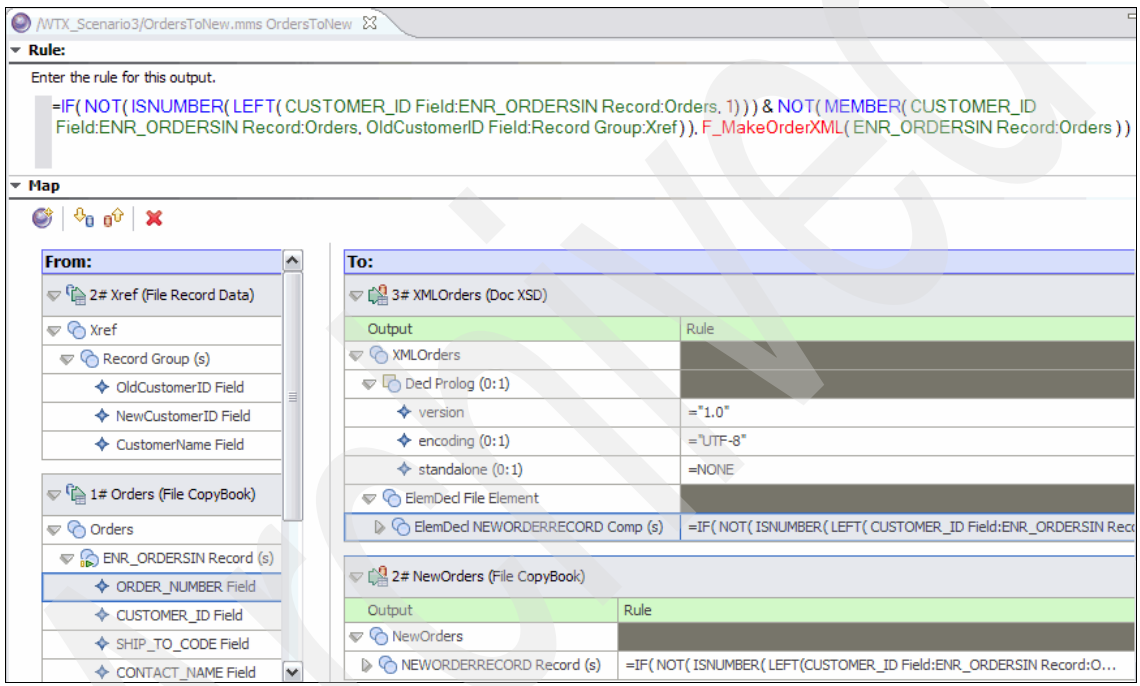

Figure 9-33 Output card #3 rules

12. Add output card #4 with the following settings (Figure 9-34):

    a. For TypeTree, type `OrderError.mtt`.
    b. For FilePath (output file), type `OrderErrors.txt`.
    c. Click **OK**.

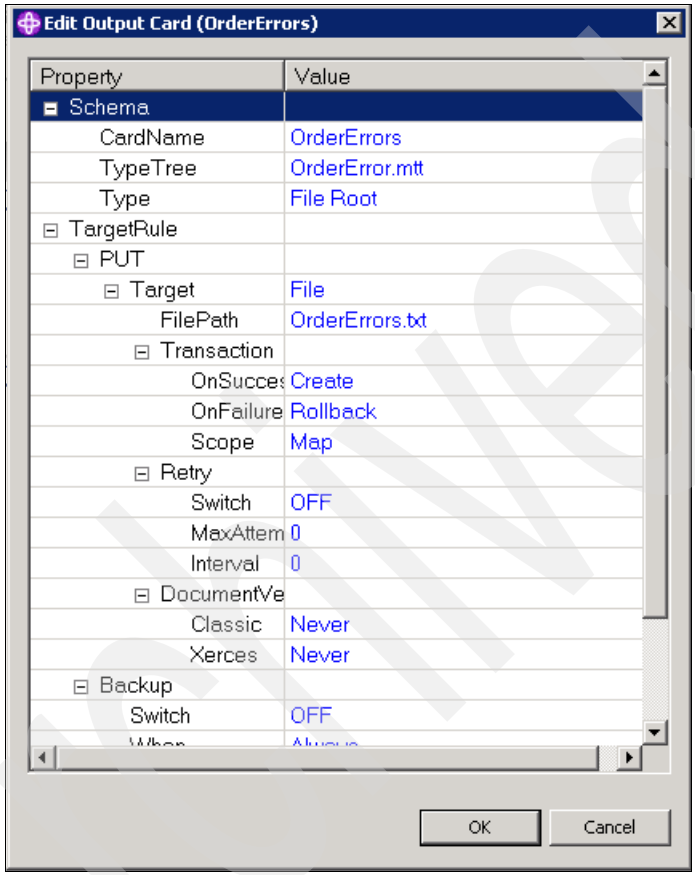

Figure 9-34 Output card #4 settings

13. Add the map rules to output card #4 (Figure 9-35).

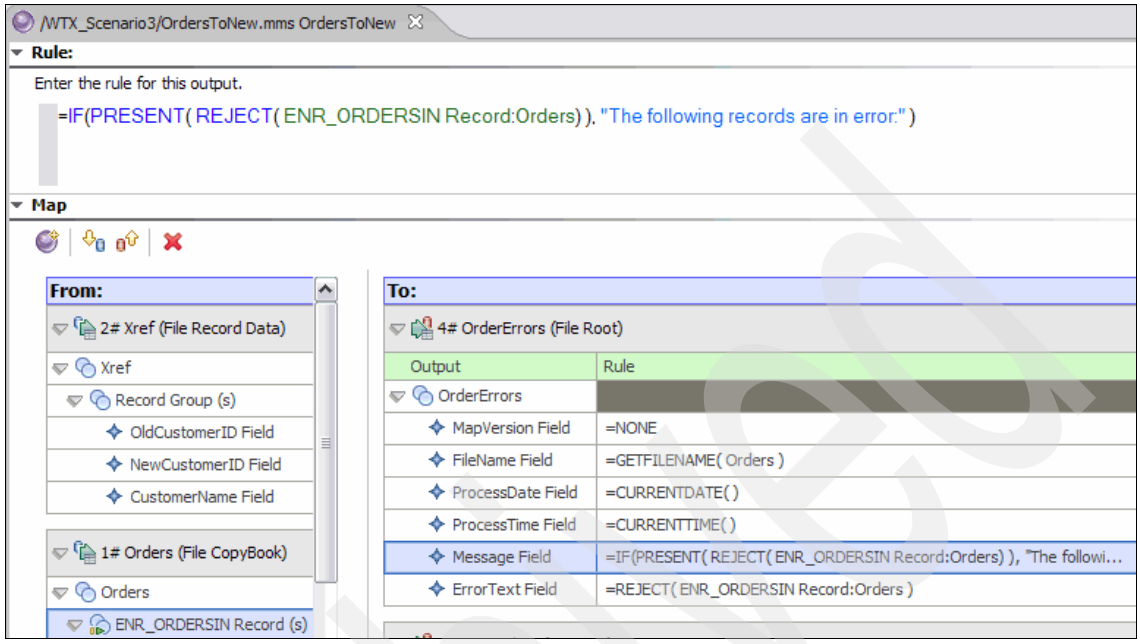

Figure 9-35 Output card #4 rules

Figure 9-36 shows the completed executable map.

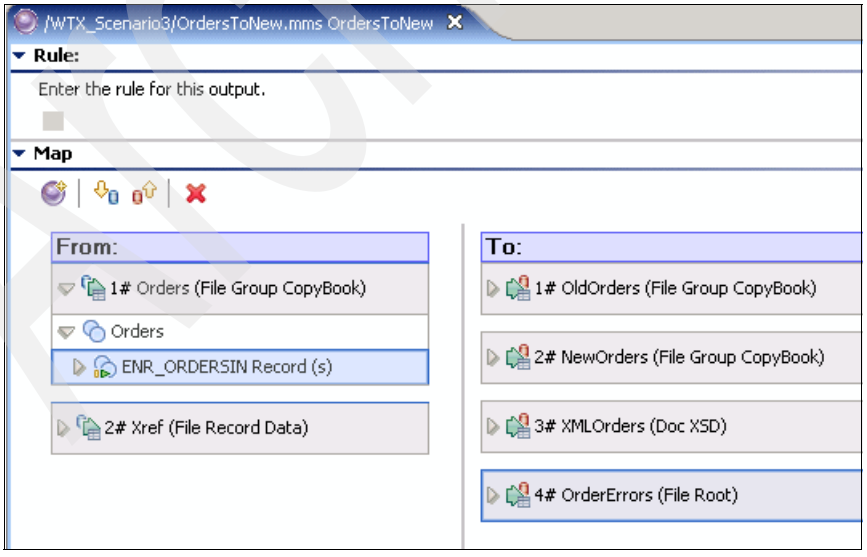

Figure 9-36 Complete OrdersToNew executable map

9.2.5 Creating the functional maps

We have two functional maps: one for creating the records for PROG3 and one for creating the XML records. We do not need a functional map to create the records for PROG2. PROG2 is part of the original architecture. Therefore, it reads the file as it is written by PROG1.

The calls to the functional maps in the map rules, described in 9.2.4, “Creating the executable map” on page 648, are used to create the functional maps with the Functional Map Wizard. Use the Functional Map Wizard to create the F_MakeOrderPROG3 (in output card #2) and F_MakeOrderXML (in output card #3) functional maps.

Tip: In the first window of the Functional Map Wizard, before you click the Create button, make sure you change the names of the input and output cards from In and Out to more meaningful names. To change the name of an input or output card, select it and click **Edit**. In the Functional Card Data window, change the name of the card.

Functional map to create the orders for PROG3

The functional map that creates the orders for PROG3, in output card #2, is called `F_MakeOrderPROG3` and has two input cards:

- Input card #1 describes the order record, as defined in the `Order.mtt` type tree.
- Input card #2 contains the NewCustomerID field, from the `Xref.mtt` type tree.

We do the lookup of the customer ID in the call to the functional map. Therefore, we do not need to do the lookup inside the map. We retrieve the NewCustomerID from the cross-referenced file based on the value of the old customer ID in the orders file.

The map rule contains the following LOOKUP command:

```
LOOKUP(NewCustomerID Field:Record Group:Xref, OldCustomerID
Field:Record Group:Xref =  CUSTOMER_ID Field:ENR_ORDERSIN
Record:Orders)
```

Open the map that was created by the Functional Map Wizard and add the map rules, as shown in Figure 9-37.

The WANT_DATE field has the following map rule:

```
=IF( PRESENT( WANT_DATE Field:OldOrder), WANT_DATE Field:OldOrder,
DATETOTEXT( ADDDAYS( TEXTTODATE( ORDER_DATE Field:OldOrder), 30 ) ) )
```

The EXTENDED_PRICE field has the following map rule:

```
=ROUND( QUANTITY_ORDERED Field:OldOrder * UNIT_PRICE Field:OldOrder, 2)
```

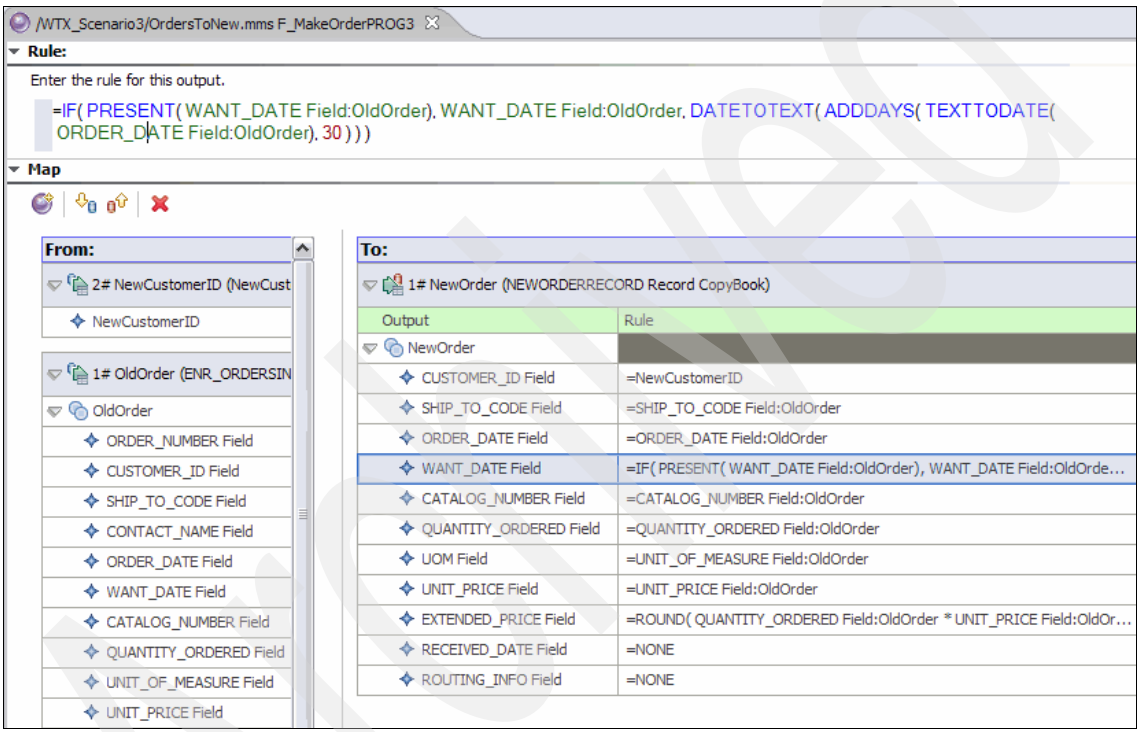

Figure 9-37 F_MakeOrderPROG3 functional map

Functional map to create the XML orders

The functional map that creates the orders for the J2EE application, in output card #3, is called `F_MakeOrderXML` and has one input card. Input card #1 describes the order record, as defined in the `Order.mtt` type tree.

Open the map created by the Functional Map Wizard and add the map rules, as shown in Figure 9-38.

The WANT_DATE field has the following map rule:

```
=IF( PRESENT( WANT_DATE Field:OldOrder), WANT_DATE Field:OldOrder,
DATETOTEXT( ADDDAYS( TEXTTODATE( ORDER_DATE Field:OldOrder), 30 ) ) )
```

The EXTENDED_PRICE field has the following map rule:

```
=ROUND( QUANTITY_ORDERED Field:OldOrder * UNIT_PRICE Field:OldOrder, 2)
```

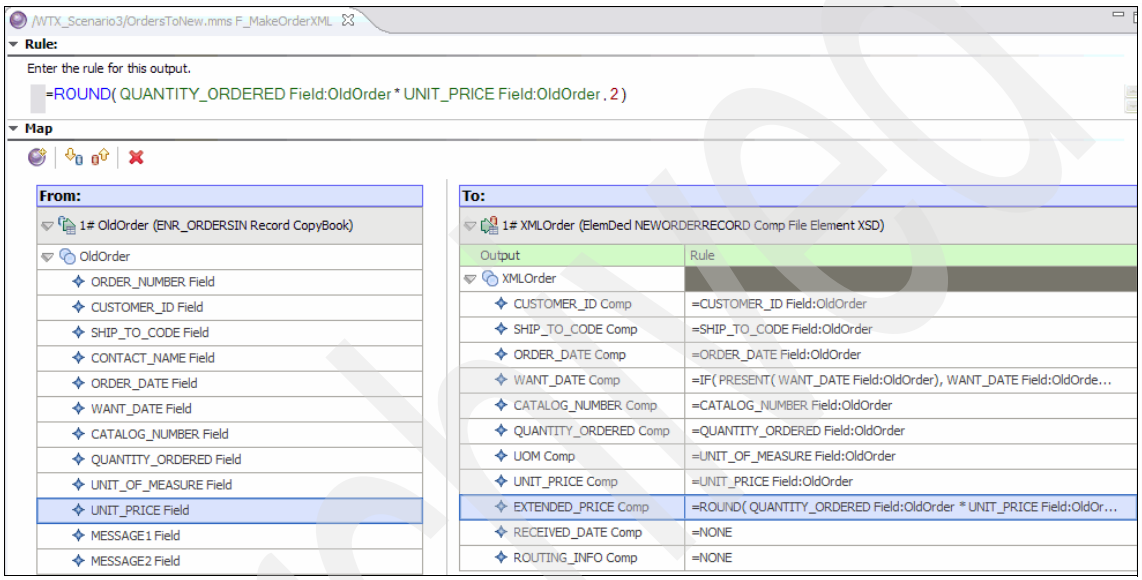

Figure 9-38 F_MakeOrderXML functional map

9.2.6 Optional: Creating the audit log map

The `Error.txt` file that we generated in output card #4 in the OrdersToNew map presents the error records as they are displayed in the input file, without any explanation of the error. We can improve this error handling in several ways. In this scenario, we create an audit log that checks for specific errors in different fields.

Creating the type tree

Since we are validating the same records that are input to OrdersToNew we can use the `Order.mtt` type tree to read the rejected records (the ones with errors) to create the audit log. The problem is that our file with rejected records contains a

header. To deal with it we need to create a new type tree, that contains all the fields in `Order.mtt` type tree and the header in the `OrderError.mtt` type tree.

If we want to use the audit log in a production environment, we do not need the header in the error file. Instead, we can just put the rejected records in a file, without a header. We can then use the `Order.mtt` type tree to read the error file and produce the audit log.

To create the `Error.mtt type` tree, we use both the `Order.mtt` and `OrderError.mtt` type trees:

1. Copy the `Order.mtt` type tree to a new one called `Error.mtt`:
   a. Right-click the **Order.mtt** type tree and select **Copy**.
   b. Right-click again and select **Paste**.
   c. Enter the new name `Error.mtt`.
   d. Click **OK**.
2. Merge the new `Error.mtt` type tree with `OrderError.mtt`:
   a. Open both type trees.
   b. Change the name of the root element of `Error.mtt` to `Root`. When merging type trees, the root element must have the same name in both trees. You can change it to something else after the merge if you want.
   c. Right-click the root element of the type tree to merge (**OrderError.mtt**) and select **Merge**.
   d. In the Merge window (Figure 9-39), go to the type tree to be merged (`Error.mtt`) and select its root element. The name of it is displayed automatically in the Merge window. Click **Merge**.

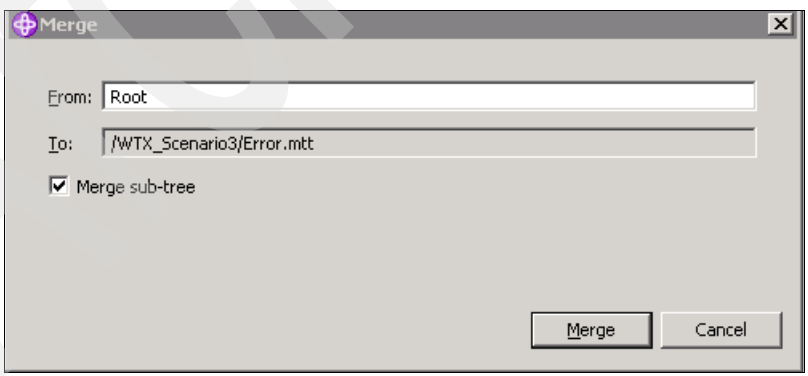

Figure 9-39 Merge type trees window

   e. After the type trees merge, click **Close**.

Tip: When merging type trees, the root element must have the same name in both trees.

Changing the type tree

Change the newly created `Error.mtt` type tree to create the audit log:

1. Open the `Error.mtt` type tree.

   For our order records, we have two validation rules. The ship-to-code field must be numeric, and the order date must precede the want date. We entered these rules when we created the Order.mtt type tree. The rules are copied into the `Error.mtt` type tree and we only need to modify.

2. Double-click the group that represents the record (**ENR_ORDERSIN**) to open it.

3. Select the **SHIP_TO_CODE** field and change the rule as follows:

   ```
   ONERROR(ISNUMBER($), "Ship-to-code must be numeric")
   ```

4. Select the **WANT_DATE** field and change the rule as follows:

   ```
   ONERROR($ > ORDER_DATE Field, "Want date must be after the order date")
   ```

   Figure 9-40 shows the rules in the record group.

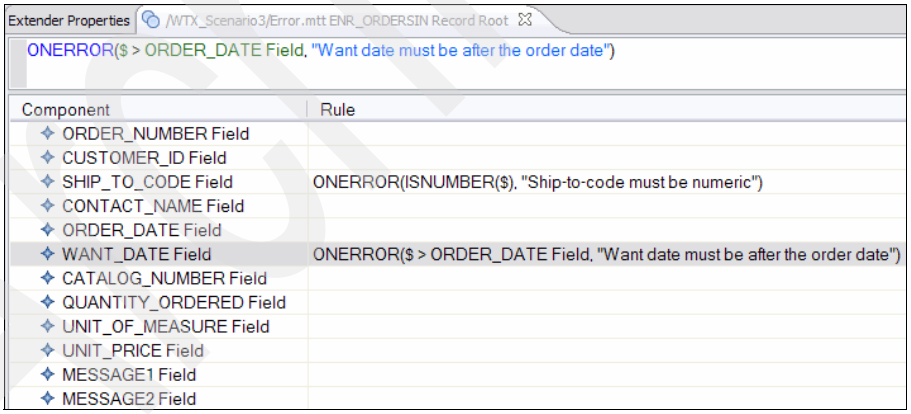

Figure 9-40 Validation rules in the Error.mtt type tree

5. Create a group called `Header`, under the Group category, that represents the header in the `OrderErrors.txt` file.

6. Double-click the **Header** group and build the component as shown in Figure 9-41.

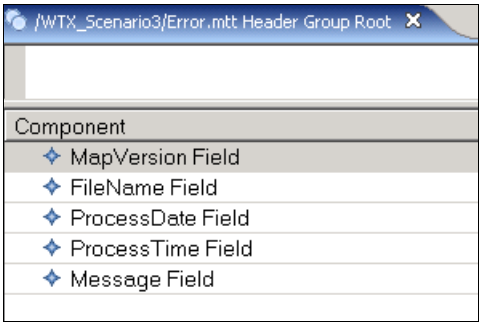

Figure 9-41 Header group in Error.mtt

7. Change the File group so that it contains the Header group before the records (Figure 9-42).

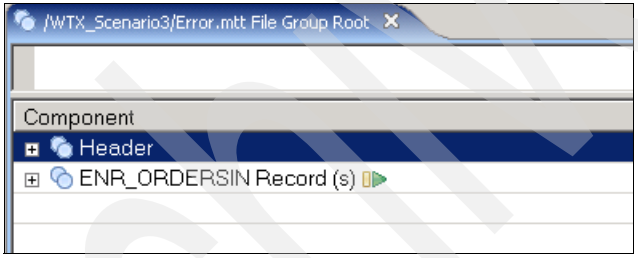

Figure 9-42 File group in the Error.mtt type tree

8. In the menu bar, select **Tree** → **Analyze** → **Structure and Logic**. Check that there are no errors. Save the type tree. Figure 9-43 shows the complete type tree.

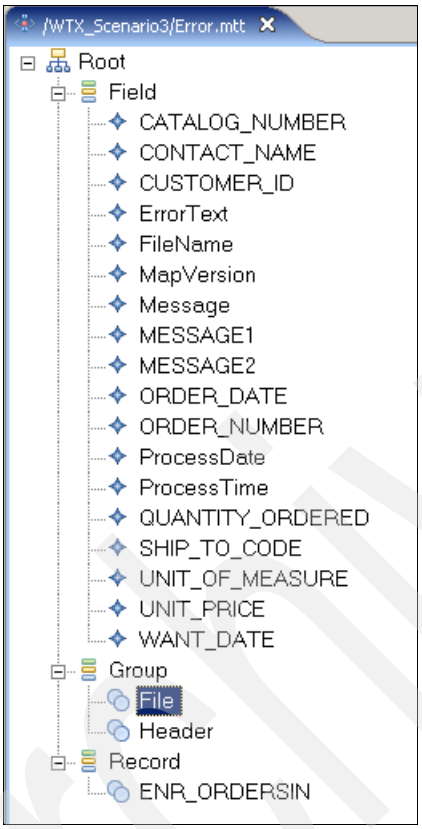

Figure 9-43 Complete Error.mtt type tree

Creating the executable map

Create a new executable map named `ErrorDetails` in the same `OrdersToNew.mms` map source file. This new map contains one input card. Input card #1 describes the error details (Figure 9-44 on page 667).

1. Create the executable map (Figure 9-44):
   a. For TypeTree, type `Error.mtt`.
   b. For FilePath (input file), type `OrderErrors.txt`.
   c. Click **OK**.

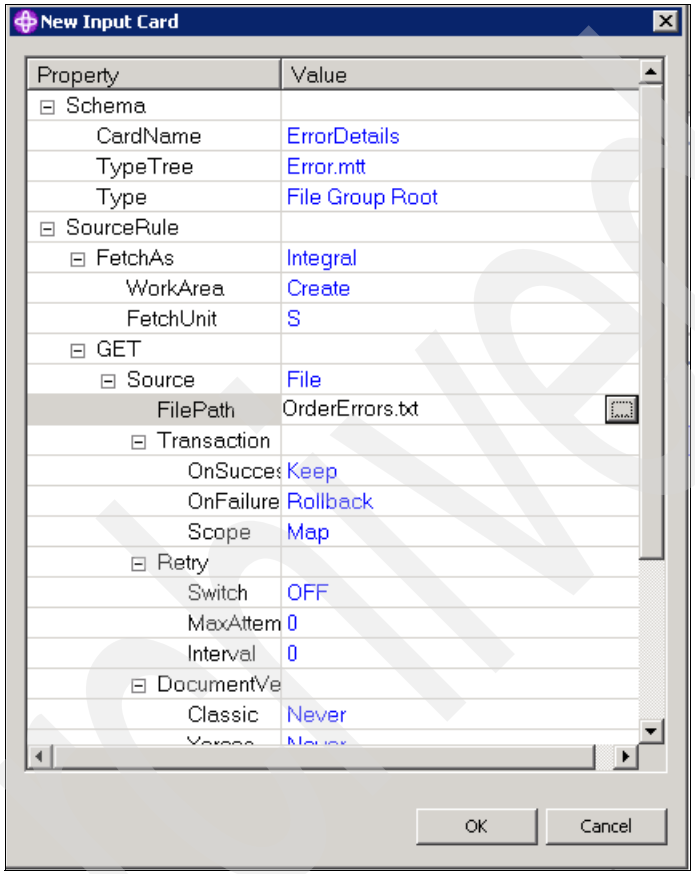

Figure 9-44 Audit log input card #1 settings

The output is written to the `ErrorDetails.log` file. Therefore, we do not need any output cards. This file can also be seen in the audit log, while you are testing your maps in Design Studio.

2. Define the audit log settings:
   a. Open the **ErrorDetails** map.
   b. From the menu bar, select **Window** → **Show View** → **Data Audit Settings**.

c. Drag the **ENR_ORDERSIN** record group to the **Audit** column of the Data Audit Settings view.

d. Change the values of Track to `Occurrence`, Details to `Error`, and Item Data to `Error` (Figure 9-45).

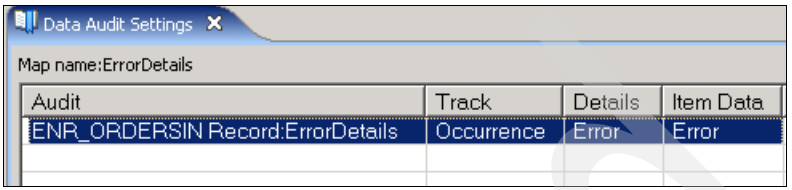

Figure 9-45 Data Audit Settings view

3. Switch on the audit log:

a. Double-click the **ErrorDetails** executable map and select **Map Settings**.

b. Change the map settings as shown in Figure 9-46.

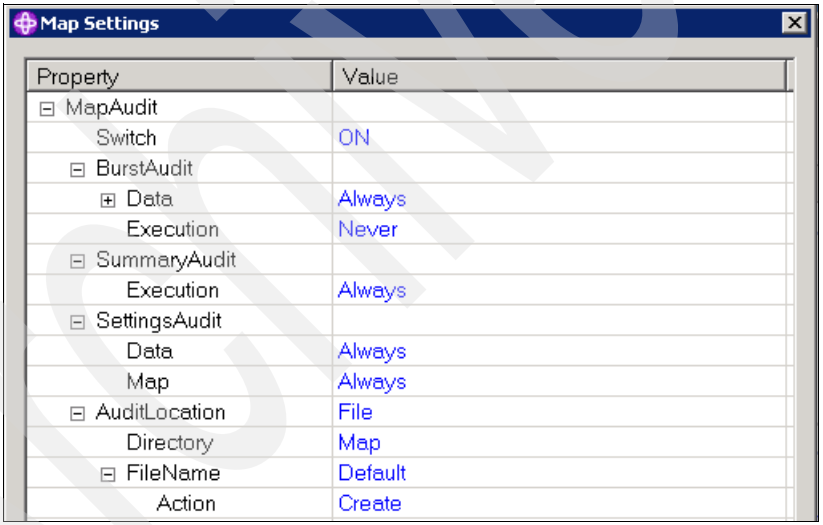

Figure 9-46 Map settings for audit log

Inserting a version in the map

In a production environment, it is not possible to know which version of the map is running. Of course, you can change the name of the compiled map to include the version number every time you deploy it. However, there is a way to have the version inside the map, without changing the name of the compiled map.

One solution is to create a type tree with one field named MapVersion and include an extra output card with this type tree in the map. In the output card, set Target to **Sink**, so that no files are produced (Figure 9-47).

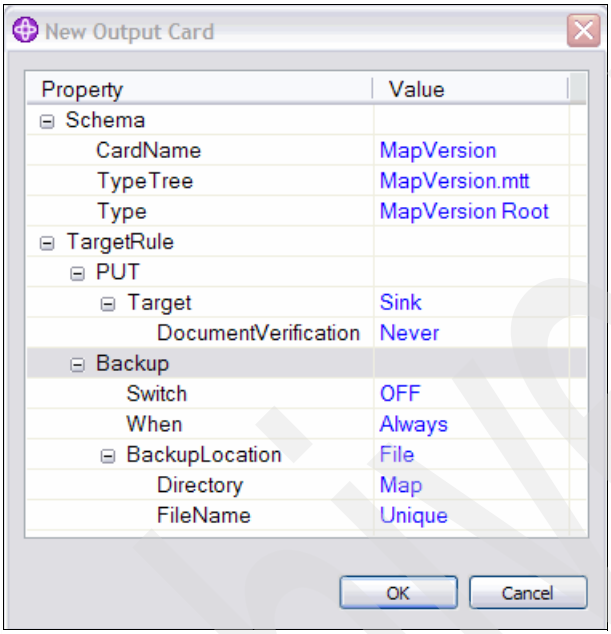

Figure 9-47 MapVersion output card settings

In the compiled map, you can search for the string "version" and check the map version. You can also include the MapVersion field in an existing type tree. In our scenario, we added the MapVersion field to the OrderError type tree.

However, because all the map rules are compiled, we are unable to see them. The only recognizable words in the compiled map are the names of the fields, initiators, terminators, and restriction lists. Therefore, in this case, we must change the type tree. In the MapVersion item, we include an initiator that is a literal (Figure 9-48):

```
Map Version<SP><SP>1.00
```

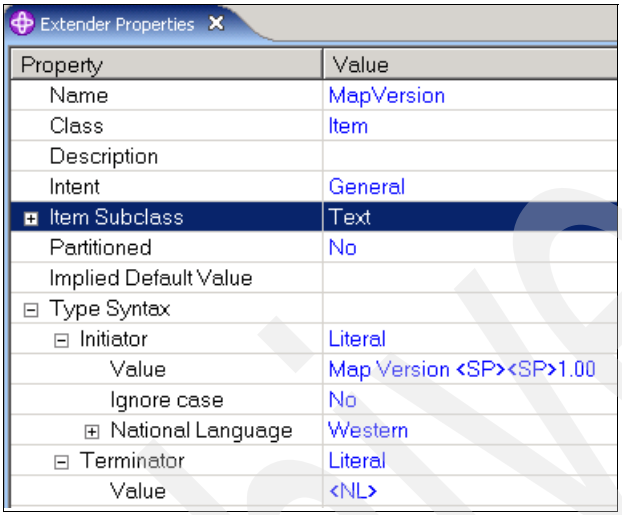

Figure 9-48 MapVersion Properties view

We must change the type tree every time that we deploy a new version of the map. It is not an optimal solution, but it is a way to read the version of the map inside the compiled code.

9.2.7 Building and running the map for unit testing

To test the map, we need the `AllOrders.txt` and `Xref.txt` input files (mentioned at the beginning of this chapter). To import these input files:

1. Right-click **WTX_Scenario3** and select **Import** → **File System**.
2. In the Import from File System window (Figure 9-49):
   a. Browse to the directory where you have your files and select the files that you want to import.
   b. The Into folder: field, ensure that it shows `WTX_Scenario3`.
   c. Click **Finish**.

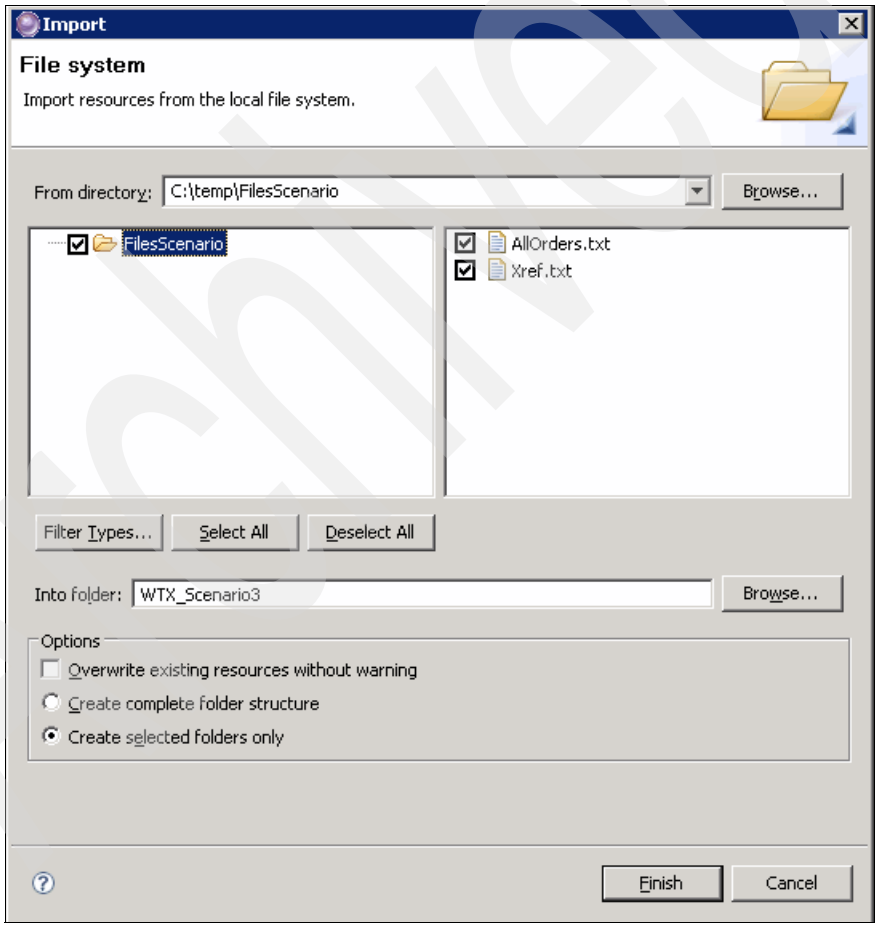

Figure 9-49 Import from File System window

The files are now in the Misc folder, under the project, as shown in Figure 9-50.

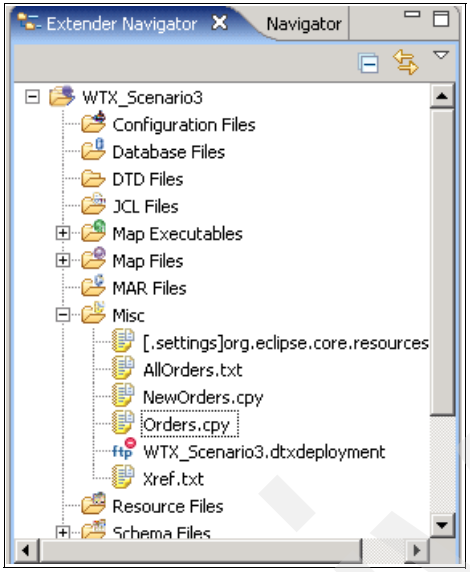

Figure 9-50 Misc folder under WTX_Scenario3

Build and test the executable map:

1. Expand the **WTX_Scenario3** project, select the folder **Map Files** and double-click **OrdersToNew.mms**. The map opens.
2. In the Outline view (Figure 9-51), double-click the **OrdersToNew** map.

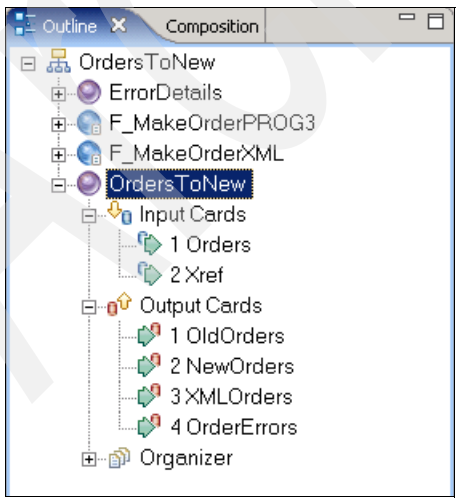

Figure 9-51 OrdersToNew.mms Outline view

3. To compile the map, from the menu bar, select **Map** → **Build**.
4. To test the map, from the menu bar, select **Map** → **Run**.

   Figure 9-52 shows the results window. Notice that there are errors. We did this on purpose because we want to show the error handling map.

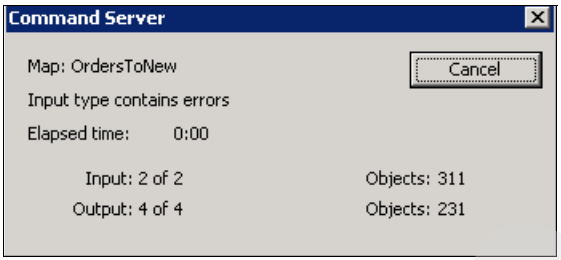

Figure 9-52 OrdersToNew results window

5. Click **Cancel** to close the Command Server window.
6. Check the results:
   a. From the menu bar, select **Map** → **View Run Results**.
   b. Select the boxes to choose which files you want to see.

The `OrderErrors.txt` file presents all the error records, but does not provide any detail about the errors. We created a new map that will read this file and create the audit log with more details about each error record. Our next task is to build and run the error handling map.

Note: There are several different ways to open, run, test, and view the results in Design Studio. We chose the one based on the menu bar for convenience of this book.

9.2.8 Building and running the error handling map

To build and run the error handling map:

1. Expand the **WTX_Scenario3** project, select the folder **Map Files** and double-click **OrdersToNew.mms**. The map opens.
2. In the Outline view (Figure 9-53), double-click the **ErrorDetails** map.

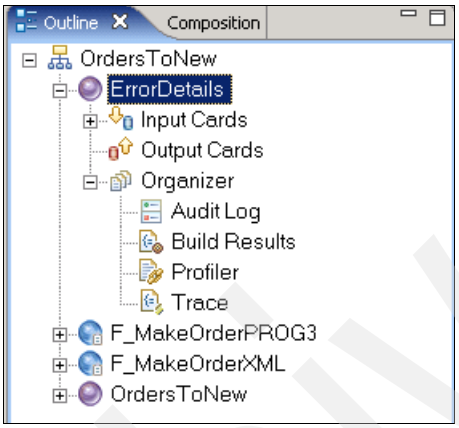

Figure 9-53 ErrorDetails.mms Outline view

3. From the menu bar, select **Map** → **Build** to compile the map.
4. To test the map, from the menu bar, select **Map** → **Run**.

   In the results window (Figure 9-54), again, notice that you get a message that the input type contains errors. Also, note that no output was generated. This is because we do not have an output card. We are generating our output as an audit log.

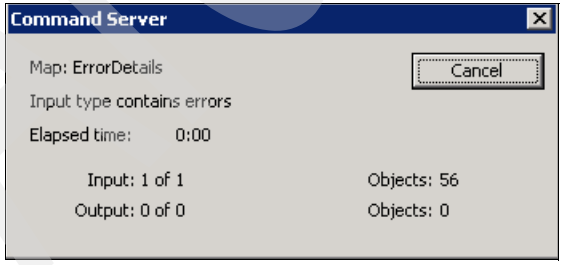

Figure 9-54 ErrorDetails results window

5. Click **Cancel** to close the results window.

6. To check the audit log:

   a. Go to the **Outline** view and expand **ErrorDetails**.
   b. Expand the **Organizer**, as shown in Figure 9-55.
   c. Double-click **Audit Log** and the audit log view opens (Figure 9-55).

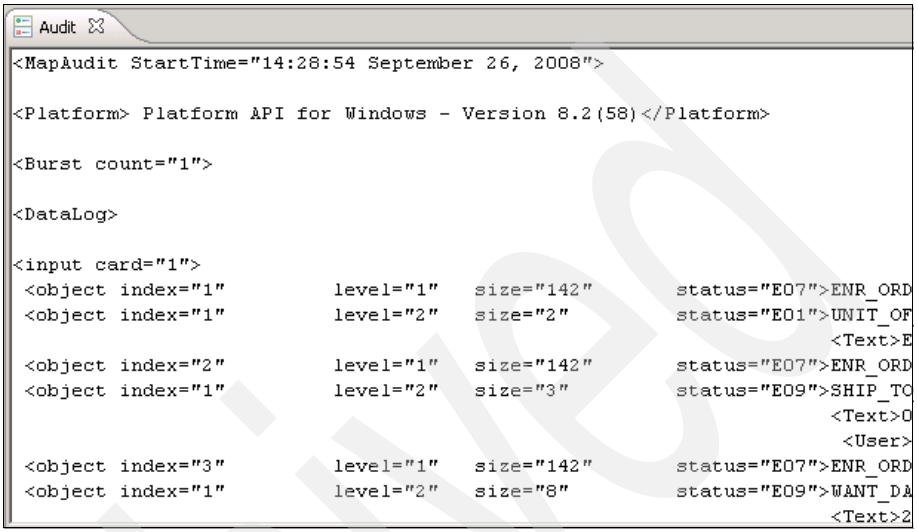

Figure 9-55 ErrorDetails audit log

You can also check the audit log file by expanding the **WTX_Scenario3** project, selecting the folder **Misc**, and double-clicking **ErrorDetails.log**.

9.2.9 Building the maps for z/OS

After testing our maps inside Design Studio, we want to run them on z/OS:

1. Build the map for the z/OS platform.
2. Expand the **WTX_Scenario3** project, select the **Map Files** folder.
3. Verify that the maps that we want to compile are there. That is, the `OrderToNew.mms` map, which contains two executable maps: `OrderToNew` and `ErrorDetails`).
4. Compile the maps:

   a. From the menu bar, select **Map** → **Build All for Specific Platform**.
   b. Select **IBM z/OS**.

5. Expand the **Map Executables** folder in the **WTX_Scenario3** project and verify that two files, `OrderToNew.mvs` and `ErrorDetails.mvs`, have been created. These files are the compiled maps to be used on z/OS.

9.2.10 Running the maps on z/OS

We assume that WebSphere Transformation Extender is correctly installed on z/OS. Besides the installation manual provided with the product, see Chapter 6, “WebSphere Transformation Extender on System z” on page 353, for hints and tips.

To upload and run the maps on z/OS:

1. Upload the maps and the necessary external files to the system. We use the FTP utility that comes with Windows to upload the files. In a command prompt window, type the following command:

   ```
   FTP -s:upload.txt myhostname
   ```

   Figure 9-56 shows the `upload.txt` file, the FTP commands to upload the maps `OrdersToNew.mvs` and `ErrorDetails.mvs`, and the `AllOrders.txt` files.

```
MYUSERNAME
MYPASSWORD
cd 'WTX.SCEN3.MAPS'
binary
quote site LRECL=80 RECFM=FB
PUT  c:\$Redbook\zOSScenario\maps\OrdersToNew.mvs ORD2NEW
PUT  c:\$Redbook\zOSScenario\maps\ErrorDetails.mvs ERRDET
ascii
cd 'WTX'
PUT  c:\$Redbook\zOSScenario\data\Xref.txt 'WTX.DATA.XREF'
quote site RECFM=VB LRECL=147 TRACKS PRIMARY=5 SECONDARY=1
PUT  c:\$Redbook\zOSScenario\data\AllOrders.txt 'WTX.DATA.TEMPORD'
quit
```

Figure 9-56 Contents of upload.txt file

   The FTP command for uploading the `AllOrders.txt` data file is preceded by a **`quote site`** statement, which gives the destination properties of the file:

   – `RECFM=VB` indicates that we used variable record length, because we want to use the `/V` parameter in the Command Server PARM line in the JCL.

   – `LRECL=147` is used because we have 143 bytes for record length + 4 bytes for variable-length type file.

   – `TRACKS, PRIMARY, SECONDARY` is used for allocation sizes of the file on the host system.

   The `AllOrders.txt` data file is used by the PROG1 COBOL program to generate the AllOrders file that is used by the WebSphere Transformation Extender maps. PROG1 simulates an application that receives orders from customers and validates them before sending the order records to the

OrdersToNew map. In our scenario, PROG1 copies the file that contains all the orders. Figure 9-57 shows the results of the upload.

```
C:\$Redbook\zOSScenario>ftp -s:uploadSCEN3_FTP.txt wtsc59oe.itso.ibm.com
Connected to wtsc59oe.itso.ibm.com.
220-FTP Server
220
User (wtsc59oe.itso.ibm.com:(none)):
331-Password:
331
230-220-FTPD1 IBM FTP CS V1R9 at wtsc59.itso.ibm.com, 19:10:17 on 2008-09-26.
230-WTXUSER is logged on.  Working directory is "/u/wtxuser".
230
ftp> cd 'WTX.SCEN3.MAPS'
250 The working directory "WTX.SCEN3.MAPS" is a partitioned data set
ftp> binary
200 Representation type is Image
ftp> quote site LRECL=80 RECFM=FB
200 SITE command was accepted
ftp> PUT  c:\$Redbook\zOSScenario\maps\OrdersToNew.mvs ORD2NEW
200 Port request OK.
125 Storing data set WTX.SCEN3.MAPS(ORD2NEW)
250 Transfer completed successfully.
ftp: 12699 bytes sent in 0.00Seconds 12699000.00Kbytes/sec.
ftp> PUT  c:\$Redbook\zOSScenario\maps\ErrorDetails.mvs ERRDET
200 Port request OK.
125 Storing data set WTX.SCEN3.MAPS(ERRDET)
250 Transfer completed successfully.
ftp: 4048 bytes sent in 0.00Seconds 4048000.00Kbytes/sec.
ftp> ascii
200 Representation type is Ascii NonPrint
ftp> cd 'WTX'
250 "WTX." is the working directory name prefix.
ftp> PUT  c:\$Redbook\zOSScenario\data\Xref.txt 'WTX.DATA.XREF'
200 Port request OK.
125 Storing data set WTX.DATA.XREF
250 Transfer completed successfully.
ftp: 133 bytes sent in 0.02Seconds 8.31Kbytes/sec.
ftp> quote site RECFM=VB LRECL=147 TRACKS PRIMARY=5 SECONDARY=1
200 SITE command was accepted
ftp> PUT  c:\$Redbook\zOSScenario\data\AllOrders.txt 'WTX.DATA.TEMPORD'
200 Port request OK.
125 Storing data set WTX.DATA.TEMPORD
250 Transfer completed successfully.
ftp: 3600 bytes sent in 0.00Seconds 3600000.00Kbytes/sec.
ftp> quit
```

Figure 9-57 Results of uploading two maps and two data files

2. Build a JCL with the call to the maps.
3. Submit the JCL.
4. Check the results in the log and the output files.

See Appendix A, “Running on z/OS” on page 679, for details about how to build and submit the JCL.

A

Running on z/OS

In this appendix, we describe the z/OS scenario implementation on the system and the way to use WebSphere Transformation Extender with Command Server.

Submitting a JCL

All necessary files, which are the compiled maps (.mvs) and data files, have already been transferred to the host. In this scenario, PROG1, PROG2, and PROG3 are simple COBOL programs that are made to simulate a real application:

- PROG1 reads the TEMPORD file and writes it as the ORDERS file, without modifications. Some outputs are produced in SYSOUT.
- PROG2 and PROG3 read the outputs that are produced by WebSphere Transformation Extender maps, calculate the amount for each line, multiplying the quantity by the unit price, count the number of lines, and sum the total amount of all lines.
  - PROG2 addresses the “old orders” file, DDNAME ORDERS2.
  - PROG3 addresses the “new orders,” DDNAME ORDERS3.

Figure A-1 shows the job control language (JCL) structure that is used in the scenario.

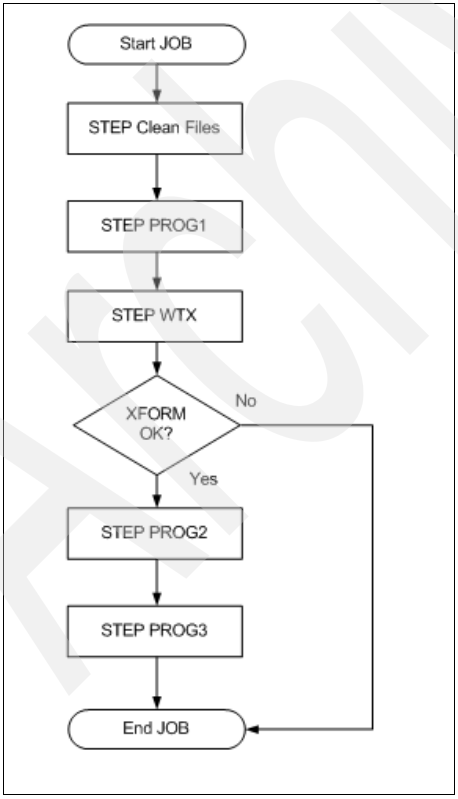

Figure A-1 JCL for scenario

Exploring the project on z/OS

To explore the project on z/OS:

1. Log on to the system (Figure A-2).

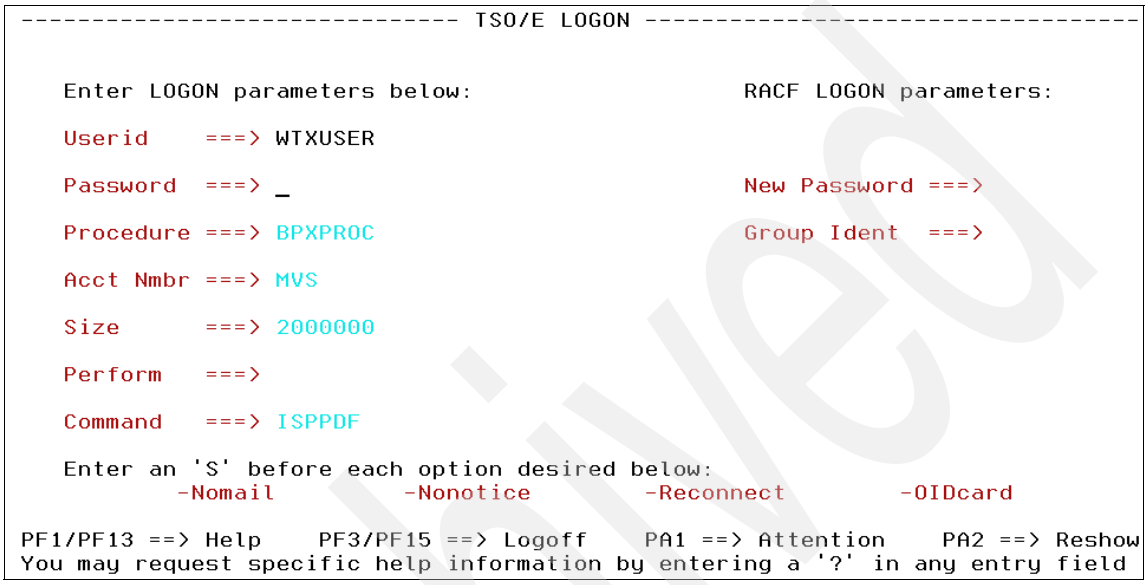

Figure A-2 Logging on to the system

2. To examine the JCL for scenario 3, enter the **3.4** command from the ISPF panel (Figure A-3).

> **Tip:** In a standard configuration of the Personal Communication tool, use the right Ctrl key to send a command.

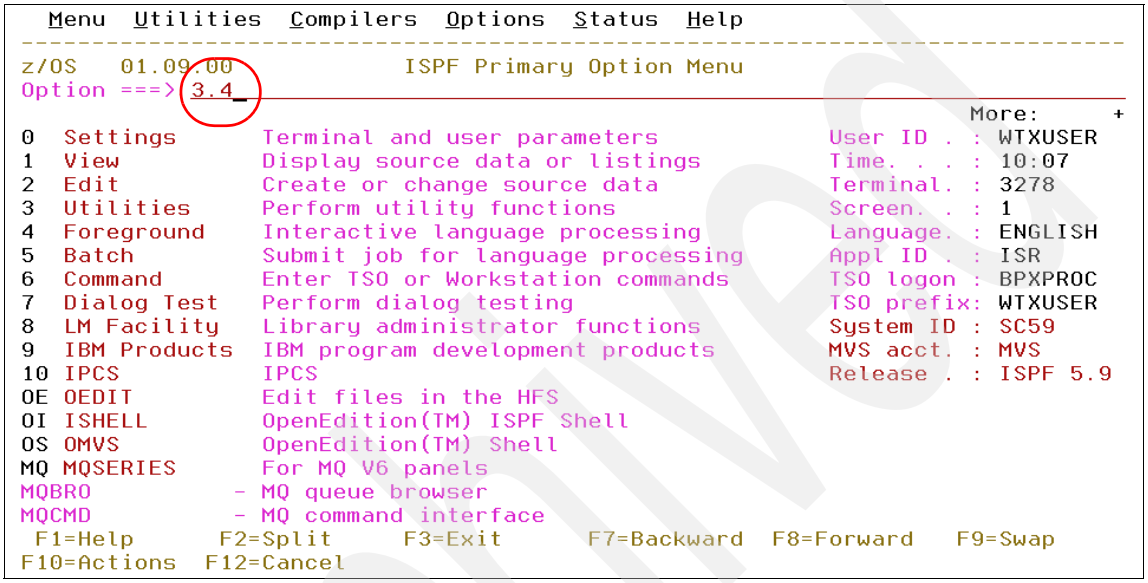

Figure A-3 ISPF - Issuing the 3.4 command

3. In the Data Set List Utility panel (Figure A-4), select the data set where the JCL is defined. In that panel, type `WTX`, which corresponds to the high qualifier of our project.

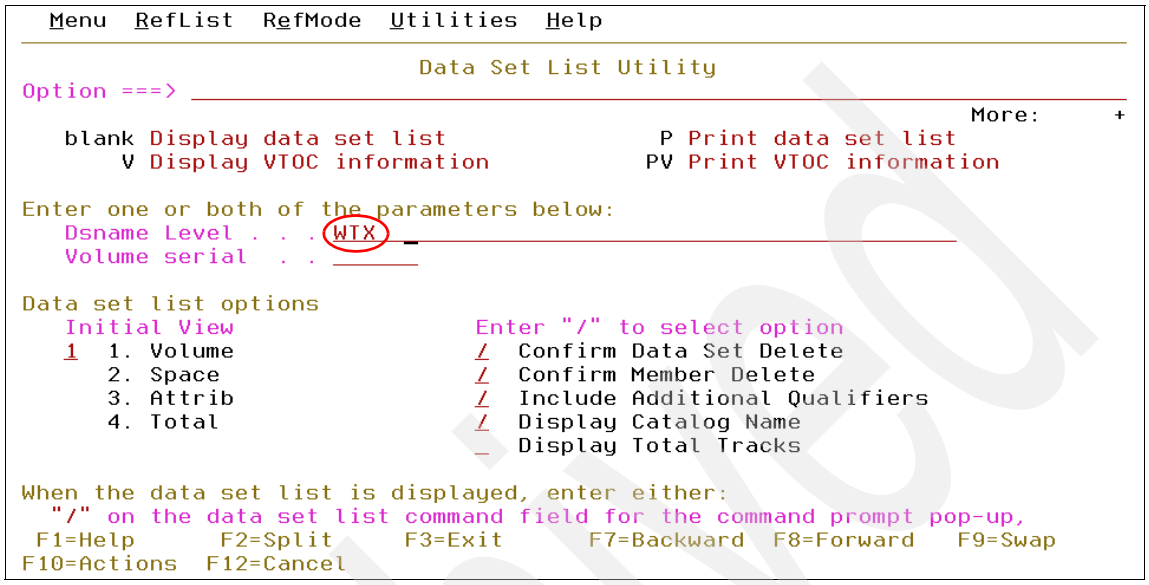

Figure A-4 Selecting the WTX high qualifier

4. In the next panel (Figure A-5), in which you see the data sets of the project, type a `b` in front of “WTX.SCEN3.JCL” and press the right Ctrl key to browse all members of this partitioned data set (PDS).

> **Tip:** To see the bottom lines, press the F8 key.

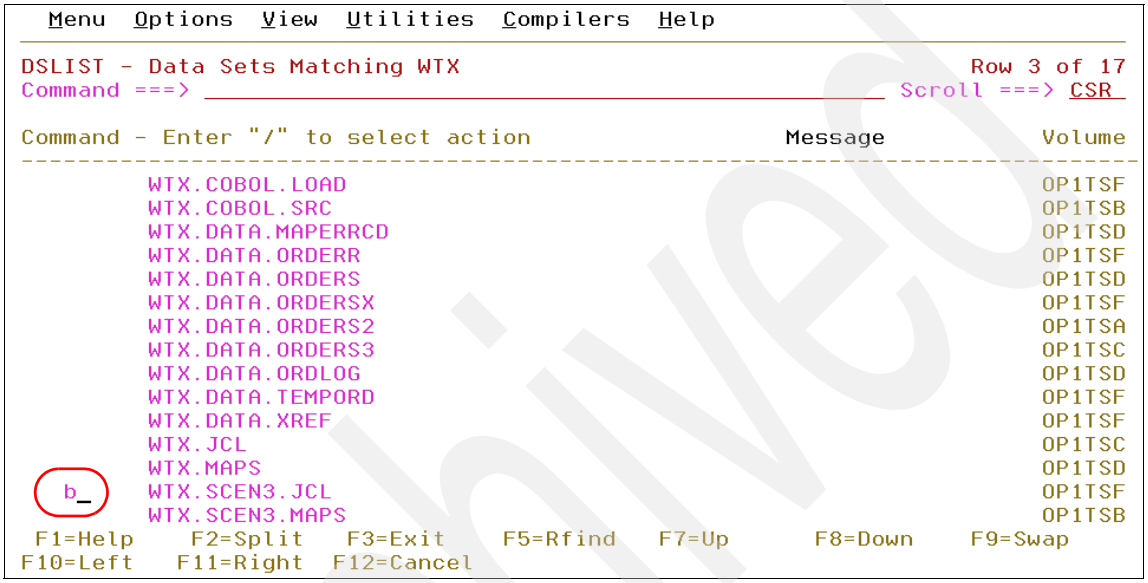

Figure A-5 List of the project's data set

5. To see and eventually modify the JCL, type an e in front of the line and press the right Ctrl key to send the command (Figure A-6).

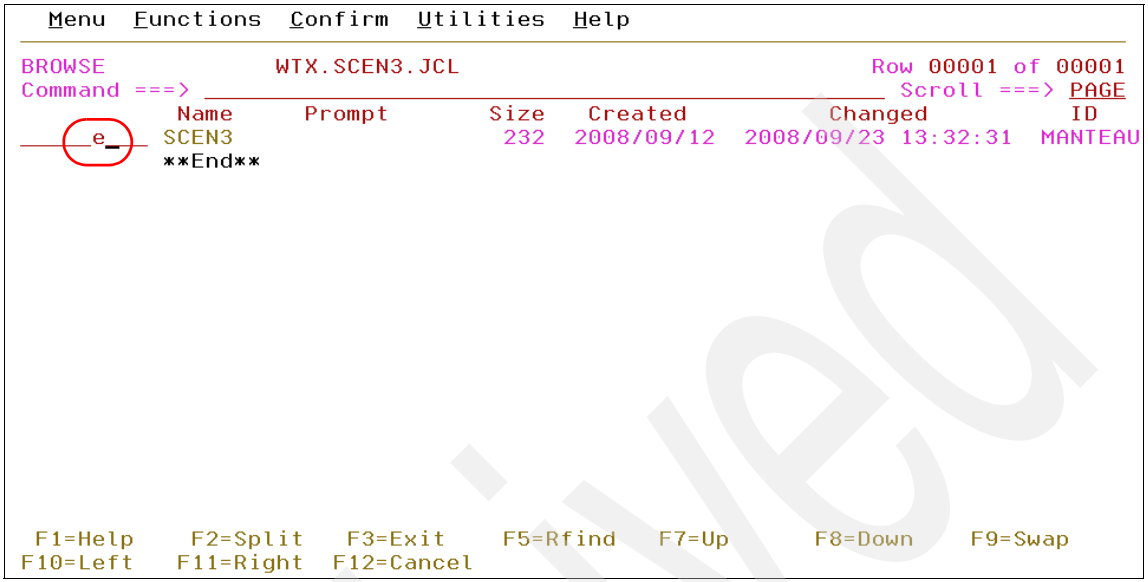

Figure A-6 Contents of the WTX.SCEN3.JCL file

We now see the JCL file and discuss the contents of this file.

Referring to Figure A-7, lines 1 and 2 are the JOB lines, for our test system.

Line 7 sets a variable with the high qualifier value for data sets.

Note: The PREFIX variable is used in all file references. Use care with the syntax of &PREFIX usage.

Lines 8 to 15 define variables for input and output files.

Line 15 sets the record-length size of temporary work files at the recommended value.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
EDIT       WTX.SCEN3.JCL(SCEN3) - 01.33               Columns 00001 00072
Command ===> _____________________________________________ Scroll ===> PAGE
****** ***************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000001 //WTXSCEN3     JOB (999,POK),'L06R',CLASS=A,REGION=0M,
000002 //             MSGCLASS=T,TIME=10,MSGLEVEL=(1,1),NOTIFY=&SYSUID
000003 //*
000004 //* ----------------- EXECUTION ENVIRONMENT ---------------------------
000005 //*        MODIFY THESE GLOBAL SYMBOLICS FOR YOUR SITE INFORMATION
000006 //*
000007 //  SET PREFIX=WTX                              <== APP PREFIX
000008 //  SET DTXLIB=BB26159.SDTXLOAD                 <== WTX LOADLIB
000009 //  SET MAPLIB=&PREFIX..SCEN3.MAPS              <== DTX MAP PDS
000010 //  SET MAPINPT=&PREFIX..DATA.ORDERS            <== INPUT FILE DSN
000011 //  SET OPROG2=&PREFIX..DATA.ORDERS2            <== OUTPUT FILE 1 DSN
000012 //  SET OPROG3=&PREFIX..DATA.ORDERS3            <== OUTPUT FILE 2 DSN
000013 //  SET OPROGXML=&PREFIX..DATA.ORDERSX          <== OUTPUT FILE 2 DSN
000014 //  SET UNIT=SYSDA                              <== WORKFILE UNIT
000015 //  SET LRECL=32760                             <== WORKFILE LRECL
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

Figure A-7 SCEN3 JCL

The next lines show the beginning of the execution steps. The execution starts with the PROG1 step in Figure A-8.

Line 17 sets the XMLLIB to the standard implementation of the XML Toolkit library.

Line 17: Line 17 is not necessary if the XML Toolkit library has been defined in the link list of the system.

Line 20 is used by the steps that launch the COBOL programs. The system must know where to fetch the executable modules of PROG1, PROG2, and PROG3.

Lines 22 to 32 are used to delete all files that are generated by the preceding executions. Output files are allocated in the JCL and, therefore, must not exist at allocation time. This step is used to delete the output files before all other tasks.

```
  File  Edit  Edit_Sett  Stop keystroke recording and save  ilities  Compilers  Test  Help
EDIT       WTX.SCEN3.JCL(SCEN3) - 01.33                    Columns 00001 00072
Command ===> _                                              Scroll ===> PAGE
000016 //  SET CLASS=*                                  <== SYSOUT CLASS
000017 //  SET XMLLIB=IXM.SIXMLOD1                      <== XML Toolkit
000018 //*
000019 //*
000020 //JOBLIB   DD DSN=&PREFIX..COBOL.LOAD,DISP=SHR
000021 //*
000022 //* ----------------- CLEAN ----------------------------------
000023 //*   Clean output files if already exists
000024 //DEL010   EXEC PGM=IDCAMS
000025 //SYSPRINT DD  SYSOUT=&CLASS
000026 //SYSIN DD *
000027   DELETE WTX.DATA.ORDERS   NONVSAM PURGE
000028   DELETE WTX.DATA.ORDERS2  NONVSAM PURGE
000029   DELETE WTX.DATA.ORDERS3  NONVSAM PURGE
000030   DELETE WTX.DATA.ORDERSX  NONVSAM PURGE
000031   DELETE WTX.DATA.ORDERR   NONVSAM PURGE
000032   DELETE WTX.DATA.ORDLOG   NONVSAM PURGE
000033   SET MAXCC = 0
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

Figure A-8 JCL page 2

The JCL starts with the PROG1 task, emulating a real application that produces a file that contains all orders types (Figure A-9 on page 688).

Lines 35 to 49 are the PROG1 step. This COBOL program reads the ORDIN (DSN WTX.DATA.TEMPORD) file, which was uploaded in a previous task. The output of PROG1 is ORDOUT (DSN WTX.DATA.ORDERS). This output file is used by WebSphere Transformation Extender as input for card 1.

ORDOUT has been allocated with RECFM=VB, because the **/V** parameter is used in the command line of WebSphere Transformation Extender. Usage of this parameter complies with the `Order.mtt` type tree, which defines <NL> as record separators.

The value of LRECL is defined as 147, which corresponds to a record length of 143 bytes and 4 bytes for VB type files. The record length of 143 bytes is defined by the `Order.mtt` type tree that is generated from the copybook that is used by PROG1 to produce the file.

The SPACE allocated to the file is defined as 1 cylinder with an extension of 1.

Note: For 3390 volumes, which are the most common disks, there are 720k per cylinder,15 tracks per cylinder,180 blocks per cylinders,12 blocks per track.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
EDIT       WTX.SCEN3.JCL(SCEN3) - 01.33                Columns 00001
Command ===> _____________________________________________ Scroll ===>
000034 //*
000035 //* ----------------- PROG 1 ----------------------------------
000036 //*              GENERATES N ORDERS
000037 //*
000038 //PROG1     EXEC PGM=PROG1
000039 //SYSPRINT DD  SYSOUT=&CLASS
000040 //SYSOUT   DD  SYSOUT=&CLASS
000041 //CEEDUMP  DD  SYSOUT=&CLASS
000042 //*
000043 //PAR      DD *
000044 0000003000000000
000045 //ORDIN    DD DSN=&PREFIX..DATA.TEMPORD,DISP=SHR
000046 //ORDOUT   DD DSN=&PREFIX..DATA.ORDERS,
000047 //            DCB=(RECFM=VB,LRECL=147,BLKSIZE=00000),
000048 //            UNIT=SYSALLDA,SPACE=(CYL,(1,1),RLSE),
000049 //            DISP=(NEW,CATLG,CATLG)
000050 //*
000051 //* ----------------- WTX   ----------------------------------
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

Figure A-9 JCL page 3

The next step is to launch WebSphere Transformation Extender Command Server (Figure A-10).

Lines 55 to 59 shows the call to WebSphere Transformation Extender. DTXCMDSRV is the module called in the &DTXLIB library (BB26159.SDTXLOAD in this system) for Command Server.

Line 59 is used to locate the XML Toolkit library, if it is not already defined in the link list of the system.

Note: The XML Toolkit library is needed even if there are no XML transformations.

Line 56 is used to put a precondition on this step. That is, it will be executed only if the result code of the step named PROG1 is less than or equal to 4 (warnings).

Line 57 is the WebSphere Transformation Extender parameter line. The map name, overrides, and options for this executable are defined in a file with the DDNAME of the CMD.

Note: The "à" character corresponds to @ in documentation as `read -@CMD`.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Iest  Help
EDIT       WTX.SCEN3.JCL(SCEN3) - 01.33                Columns 00001 00072
Command ===> _                                           Scroll ===> PAGE
000052 //*              TRANSFORM THE ORDERS
000053 //*            DISPATCH ORDERS TO SYSTEMS
000054 //*
000055 //DTX    EXEC PGM=DTXCMDSV,REGION=0M,
000056 //            COND=(4,LE,PROG1),
000057 //            PARM=' "-àCMD"'              <== COMMAND LINE IN FILE
000058 //STEPLIB  DD  DISP=SHR,DSN=&DTXLIB
000059 //           DD  DISP=SHR,DSN=&XMLLIB
000060 /*
```

Figure A-10 JCL page 4 - WTX command step

Figure A-11 on page 691 shows the CMD file. The CMD file in line 62 is a JCL inline file. It ends with /* in line 74. The content is picked up by WebSphere Transformation Extender as a command line when starting execution.

In this file, two maps are called in sequence: first MYMAP1 and then MYMAP2. The name of the map must be first, followed by all options for that map. The options can be in any order.

Syntax: All options must start with a forward slash (/) or minus sign (-). Otherwise, WebSphere Transformation Extender takes it as a map name, starts the transformation, and fails.

The `/TS` on line 64 indicates to generate a trace file with summary only. This overrides any trace settings that are set during development.

Tip: If settings for trace are ON during development, the map is built with this setting, and no /T parameter is given in JCL, a trace file is produced that corresponds to the settings. To set trace to OFF, use `/T`.

The `/Z28` on line 64 is used to catch warning code 28 that is issued by WebSphere Transformation Extender during input data validation. WebSphere Transformation Extender code 28 corresponds to the "`Input type contains errors`" message. It sets a JCL return code level 04 if nothing is done. Using `/Z28` inhibits this warning code for the MYMAP1 map and sets the return code for the step to 0. This mechanism was used because the input data errors are handled by the map itself, using the restart flag and Reject function features.

In the JCL, PROG2 and PROG3 are launched only if the return code from WebSphere Transformation Extender is 0.

The `/I1` on line 65 overrides the name of the first card input file to TXIN. `/VX15 TXIN` indicates to add a <NL> at each end of record for that file. Line 66 does the same for input card 2, file XREF.

Note: In line 66, `/I2 XREF` was not necessary, because the file name defined in the card is `xref.txt`. WebSphere Transformation Extender might have derived XREF as the default DDNAME.

Lines 67 to 70 override file names for output cards 1 to 4. Because the files are Windows compatible, a new-line character is added at the end of each record, using the `/VX15` option. For card 3, because XML is generated, there is no need to add line separators. The XML tags that are produced fill each record as much as possible, then fill the next record, and so on.

Tip: When using XML data, use large record length files for better performance.

Line 73 defines the file of the first input card of MYMAP2. Notice that the file that was used has the same DDNAME as the fourth output card of MYMAP1. It reads the results of MYMAP1.

Note: Chaining maps is easy in JCL.

MYMAP2 map does not have any output card. The audit log file that describes input validation errors is the only output.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
EDIT       WTX.SCEN3.JCL(SCEN3) - 01.33               Columns 00001 00072
Command ===> _____________________________________________ Scroll ===> CSR
000061 //*   WTX COMMAND LINE 'CMD' FILE
000062 //CMD DD *
000063 MYMAP1
000064 /TS  /Z28
000065 /I1 TXIN      /VX15 TXIN
000066 /I2 XREF      /VX15 XREF
000067 /O1 OPROG2    /VX15 OPROG2
000068 /O2 OPROG3    /VX15 OPROG3
000069 /O3 OPROGXML
000070 /O4 ORDERR    /VX15 ORDERR
000071 MYMAP2
000072 /T /Z28
000073 /I1 ORDERR    /VX15 ORDERR
000074 /*
000075 //*
```

Figure A-11 PARAMS file for WTX

For the WebSphere Transformation Extender step, system output files must be defined (Figure A-12).

Line 76 to 78 send system outputs to SYSOUT.

Line 80 to 81 associate the DDNAME of the maps to physical files (DSN). The maps were uploaded as members of a PDS.

Line 83 associates the predefined DDNAME for WebSphere Transformation Extender logs with SYSOUT.

Line 84 is commented, because the audit log file will be associated to a specific file.

Line 85 defines the WebSphere Transformation Extender trace file.

Lines 88 and 89 associate the DDNAMEs of the two input cards of MYMAP1 to the physical files.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Iest  Help
EDIT       WTX.SCEN3.JCL(SCEN3) - 01.35               Columns 00001 00072
Command ===> _____________________________________________ Scroll ===> CSR
000074 /*
000075 //*
000076 //SYSPRINT DD  SYSOUT=&CLASS
000077 //SYSOUT   DD  SYSOUT=&CLASS
000078 //CEEDUMP  DD  SYSOUT=&CLASS
000079 //*
000080 //MYMAP1   DD  DISP=SHR,DSN=&MAPLIB(ORD2NEW)
000081 //MYMAP2   DD  DISP=SHR,DSN=&MAPLIB(ERRDET)
000082 //*
000083 //DTXLOG   DD  SYSOUT=&CLASS            Execution log
000084 //*DTXAUD   DD  SYSOUT=&CLASS            Audit file
000085 //DTXTRCE  DD  SYSOUT=&CLASS            Trace file
000086 //*
000087 //* Define the input datasets
000088 //TXIN     DD  DISP=SHR,DSN=&MAPINPT
000089 //XREF     DD  DISP=SHR,DSN=&PREFIX..DATA.XREF
000090 //*
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

Figure A-12 JCL page 6 - System, maps, and input data files

The card 4 file of MYMAP1 and that audit log file for MYMAP2 must be allocated (Figure A-13). OPROG2, OPROG3, and OPROG7 are the DDNAMEs for output cards 1, 2, and 3 of MYMAP1. They are associated with corresponding variables that are set at the beginning of the JCL.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
EDIT       WTX.SCEN3.JCL(SCEN3) - 01.33                Columns 00001 00072
Command ===> _                                          Scroll ===> CSR
000092 //* Define the output datasets
000093 //OPROG2   DD  DSN=&OPROG2,
000094 //             DCB=(RECFM=V,LRECL=147),
000095 //             UNIT=&UNIT,
000096 //             SPACE=(CYL,(5,1),RLSE),
000097 //             DISP=(NEW,CATLG,DELETE)
000098 //*
000099 //OPROG3   DD  DSN=&OPROG3,
000100 //             DCB=(RECFM=V,LRECL=87),
000101 //             UNIT=&UNIT,
000102 //             SPACE=(CYL,(5,1),RLSE),
000103 //             DISP=(NEW,CATLG,DELETE)
000104 //*
000105 //OPROGXML DD  DSN=&OPROGXML,
000106 //             DCB=(RECFM=V,LRECL=256),
000107 //             UNIT=&UNIT,
000108 //             SPACE=(CYL,(15,1),RLSE),
000109 //             DISP=(NEW,CATLG,DELETE)
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap      F10=Left     F11=Right    F12=Cancel
```

Figure A-13 JCL page 7 - Output file allocations

Figure A-14 shows the allocation for the Errors file (output card 4 of MYMAP1) and for the audit log file. In the scenario, MYMAP1-ORD2NEW writes, in the ORDERR file, all rejected data. MYMAP1 has no audit log settings turned on. The file ORDERR is then read by MYMAP2 with the audit log settings turned on. The result of audit log is written in the DTXAUD file.

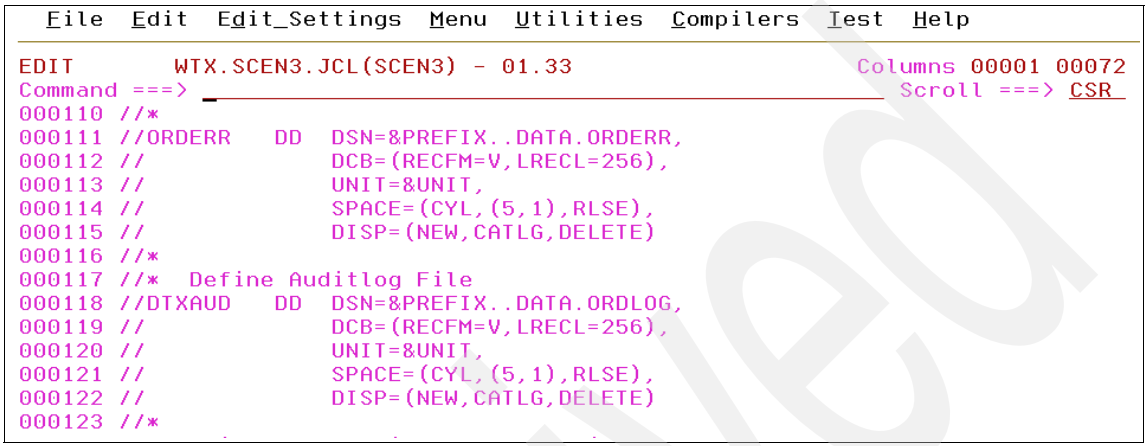

Figure A-14 JCL page 8 - Error file and audit log file

All cards of all maps have a DDNAME that is defined and associated with a DSN. The temporary work files must now be defined (Figure A-15).

In MYPAP1, there are two input cards and four output cards. This means that WebSphere Transformation Extender might use the following equation:

```
1 + (2 x 2 inputs) + (2 x 4 outputs) = 13 temporary files
```

Being in the same execution step, MYMAP2 reuses the temporary files of MYMAP1.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Iest  Help
EDIT       WTX.SCEN3.JCL(SCEN3) - 01.33                Columns 00001
Command ===> _____________________________________________ Scroll ===>
000123 //*
000124 //* Define the Static temporary file for WTX
000125 //* The maximum number of temporary files used by a map is
000126 //* (2 * number of inputs) + (2 * number of outputs) + 1
000127 //* Minimum is number of inputs + number of outputs +1
000128 //*
000129 //SYSTMP01 DD DSN=&&TEMP01,
000130 //            DISP=(NEW,DELETE,DELETE),  .
000131 //            DCB=(RECFM=FBS,LRECL=&LRECL),
000132 //            UNIT=&UNIT,
000133 //            SPACE=(CYL,(5,1))
000134 //SYSTMP02 DD DSN=&&TEMP02,
000135 //            DISP=(NEW,DELETE,DELETE),
000136 //            DCB=(RECFM=FBS,LRECL=&LRECL),
000137 //            UNIT=&UNIT,
000138 //            SPACE=(CYL,(5,1))
000139 //SYSTMP03 DD DSN=&&TEMP03,
000140 //            DISP=(NEW,DELETE,DELETE),
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap      F10=Left     F11=Right    F12=Cancel
```

Figure A-15 JCL page 9 - Temporary files for WTX

Figure A-16 shows the definition of the last files.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
EDIT       WTX.SCEN3.JCL(SCEN3) - 01.37                Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR
000178 //             SPACE=(CYL,(5,1))
000179 //SYSTMP11 DD DSN=&&TEMP11,
000180 //             DISP=(NEW,DELETE,DELETE),
000181 //             DCB=(RECFM=FBS,LRECL=&LRECL),
000182 //             UNIT=&UNIT,
000183 //             SPACE=(CYL,(5,1))
000184 //SYSTMP12 DD DSN=&&TEMP12,
000185 //             DISP=(NEW,DELETE,DELETE),
000186 //             DCB=(RECFM=FBS,LRECL=&LRECL),
000187 //             UNIT=&UNIT,
000188 //             SPACE=(CYL,(5,1))
000189 //SYSTMP13 DD DSN=&&TEMP13,
000190 //             DISP=(NEW,DELETE,DELETE),
000191 //             DCB=(RECFM=FBS,LRECL=&LRECL),
000192 //             UNIT=&UNIT,
000193 //             SPACE=(CYL,(5,1))
000194 //*
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

Figure A-16 JCL page 10 - End of the temporary files allocation

After the WTX step, PROG2 and PROG3 steps statements must follow. They are embedded in an IF-ENDIF structure. These steps are executed only if the return code of the WTX step is less than or equal to 4 (Figure A-17).

Line 201 is a test to check if the preceding steps were only warnings (RC<=4). If they were not a warning, execution stops. (The “ENDIF” is last statement of the JCL.) If the condition is satisfied, then the embedded steps in IF-ENDIF are executed.

Lines 205 to 212 define the PROG2 step. PROG2 is a COBOL program. The executable module is located by the JOBLIB statement at the beginning of JCL. PAR is a JCL inline file that is used by PROG2.

ORDIN2 is the DDNAME of the `WTX.DATA.ORDERS2` input file, which is generated by the WebSphere Transformation Extender map `MYMAP1 -ORD2NEW`.

There is no output file. Therefore, the program displays all information in SYSOUT.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
EDIT       WTX.SCEN3.JCL(SCEN3) - 01.33                Columns 00001 00072
Command ===> _____________________________________________ Scroll ===> CSR
000199 //*
000200 //* ============ TRANSFORMATION OK ============================
000201 //COND01  IF RC <= 4 THEN
000202 //*
000203 //*         ---------- PROG 2 ------------------------------------
000204 //*              OLD ORDERS MANAGEMENT
000205 //PROG2     EXEC PGM=PROG2
000206 //SYSPRINT  DD SYSOUT=*
000207 //SYSOUT    DD SYSOUT=*
000208 //*
000209 //PAR       DD *
000210 0000003000000000
000211 //ORDIN2    DD DSN=&OPROG2,DISP=SHR
000212 //*
000213 //*
000214 //*
000215 //*         ---------- PROG 3 ------------------------------------
000216 //*              NEW ORDERS MANAGEMENT
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

Figure A-17 JCL page 11 - PROG2 step

The next step is PROG3 (Figure A-18). PROG3 works the same way as PROG2. Line 226 is the END statement for the IF in line 201.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Iest  Help
EDIT       WTX.SCEN3.JCL(SCEN3) - 01.36                Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR
000214 //*
000215 //*        ---------- PROG 3 ------------------------------------
000216 //*              NEW ORDERS MANAGEMENT
000217 //PROG3     EXEC PGM=PROG3
000218 //SYSPRINT  DD SYSOUT=*
000219 //SYSOUT    DD SYSOUT=*
000220 //*
000221 //PAR       DD *
000222 0000003000000000
000223 //ORDIN3    DD DSN=&OPROG3,DISP=SHR
000224 //*
000225 //*
000226 //         ENDIF
000227 //* ------------------------------------------------------------
000228 //
****** **************************** Bottom of Data ****************************

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

Figure A-18 JCL page 12 - PROG3 step

This is the end of JCL.

Submitting the job and reviewing the job results

To submit the job:

1. Type SUBMIT or SUB in the command line of the EDIT panel (Figure A-19).

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Iest  Help
EDIT       WTX.SCEN3.JCL(SCEN3) - 01.34                Columns 00001 00072
Command ===> submit_                                    Scroll ===> CSR
****** ***************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000001 //WTXSCEN3     JOB (999,POK),'L06R',CLASS=A,REGION=0M,
000002 //             MSGCLASS=T,TIME=10,MSGLEVEL=(1,1),NOTIFY=&SYSUID
000003 //*
000004 //* ----------------- EXECUTION ENVIRONMENT --------------------------
000005 //*         MODIFY THESE GLOBAL SYMBOLICS FOR YOUR SITE INFORMATION
000006 //*
000007 //   SET PREFIX=WTX                                <== APP PREFIX
000008 //   SET DIXLIB=BB26159.SDIXL06D                   <== WTX L06DLIB
```

Figure A-19 Submitting the JOB

A confirmation with a JOB number is displayed at the bottom of panel (Figure A-20).

```
IKJ56250I JOB WTXSCEN3(JOB15418) SUBMITTED
***  _
```

Figure A-20 JOB submission

2. Press the right Ctrl key to send the command.

   If there are any errors in execution, you see a message such as the example in Figure A-21.

```
09.53.22 JOB15418 $HASP165 WTXSCEN3 ENDED AT WTSC59 MAXCC=12 CN(INTERNAL
***  _
```

Figure A-21 Job results - Execution error message

If the error message is a JCL error, the message will indicate that it is a JCL error. If there is no error, you see a message like the example in Figure A-22.

```
15.11.50 JOB15433 $HASP165 WTXSCEN3 ENDED AT WTSC59 MAXCC=0
***  _
```

Figure A-22 Job results - No errors

3. Press the right Ctrl key to send the command.
4. To see the SYSOUT and examine the results while keeping the present panel, create a new session. Place the cursor on the first line of the panel menu line and press the F2 key. Press the F9 key if you want to switch from one panel to the other, alternatively.

5. To see the results of the execution, in the command line of the new panel, type the following command (Figure A-23):

   ```
   TSO SDSF
   ```

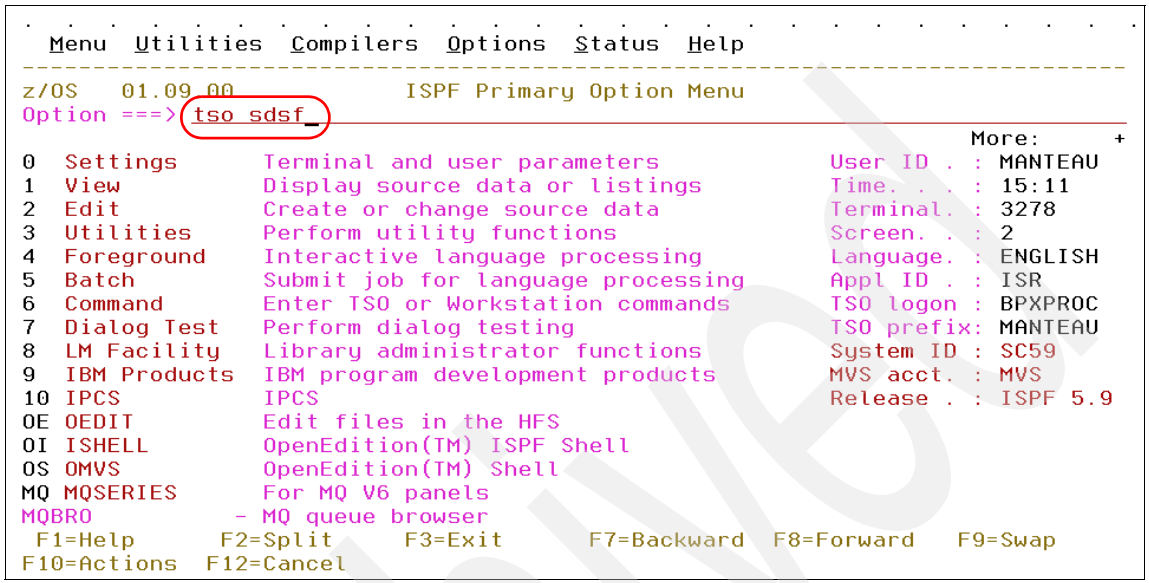

Figure A-23 OB results - Switching to the SDSF panels

6. In the SDSF panel, to select the JOBs among all jobs in the submission queues of the system, filter the name by typing a PREFIX command. For example, type the following command to select all jobs with a name that starts with WTX (Figure A-24):

   ```
   PRE WTX*
   ```

   The first line of the JCL starts with the name of the job, which is `//WTXSCEN3`.

```
HQX7740 ----------------  SDSF PRIMARY OPTION MENU  -------------------------
COMMAND INPUT ===> pre WTX*_                                 SCROLL ===> PAGE

DA     Active users                          INIT   Initiators
I      Input queue                           PR     Printers
O      Output queue                          PUN    Punches
H      Held output queue                     RDR    Readers
ST     Status of jobs                        LINE   Lines
                                             NODE   Nodes
LOG    System log                            SO     Spool offload
SR     System requests                       SP     Spool volumes
MAS    Members in the MAS
JC     Job classes                           RM     Resource monitor
SE     Scheduling environments               CK     Health checker
RES    WLM resources
ENC    Enclaves                              ULOG   User session log
PS     Processes

END    Exit SDSF
```

Figure A-24 JOB results - Setting prefix name for the queue

7. In the same panel, type `O` (or `H`) and press the right Ctrl key to see the selected job results.
8. In front of our job output, which was selected with the Job ID given when it was submitted, type a question mark (?) to see more details (Figure A-25).

```
SDSF OUTPUT ALL CLASSES ALL FORMS     LINES 797         LINE 1-1 (1)
COMMAND INPUT ===>                                        SCROLL ===> PAGE
NP    JOBNAME  JobID    Owner    Prty C Forms    Dest              Tot-Rec
 ?_   WTXSCEN3 JOB15433 MANTEAU   144 T STD      LOCAL                 797
```

Figure A-25 Job results

In the next panel (Figure A-26), the three first lines (StepName JES2) provide information about the execution, the JCL, file allocation, DDNAME resolution, warnings or errors in execution, and so on. The DEL010 line contains information about the “Clean file step”.

The PROG1 line contains information about the execution of the PROG1 step. Type an s in front of the PROG1 line and press the right Ctrl key to send it as shown in Figure A-26.

```
SDSF JOB DATA SET DISPLAY - JOB WTXSCEN3 (JOB15433)    LINE 1-9 (9)
COMMAND INPUT ===>                                      SCROLL ===> PAGE
NP    DDNAME   StepName ProcStep DSID Owner    C Dest          Rec-Cnt Page
      JESMSGLG JES2                 2 MANTEAU  T LOCAL              22
      JESJCL   JES2                 3 MANTEAU  T LOCAL             272
      JESYSMSG JES2                 4 MANTEAU  T LOCAL             182
      SYSPRINT DEL010             106 MANTEAU  T LOCAL              29
 s_   SYSOUT   PROG1              108 MANTEAU  T LOCAL              30
      DTXLOG   DTX                113 MANTEAU  T LOCAL             215
      DTXTRCE  DTX                114 MANTEAU  T LOCAL              15
      SYSOUT   PROG2              116 MANTEAU  T LOCAL              12
      SYSOUT   PROG3              118 MANTEAU  T LOCAL              20
```

Figure A-26 JOB results - Details for each step

The PROG1 COBOL program generates, in SYSOUT, some information for each order line and, at the end, the number of orders (Figure A-27).

```
SDSF OUTPUT DISPLAY WTXSCEN3 JOB15433  DSID   108 LINE 9       COLUMNS 02- 81
COMMAND INPUT ===> _                                         SCROLL ===> CSR
XRF-1A-150 ref for Customer ABCWIDGET
BDQ-4F-190 ref for Customer ABCWIDGET
ABC-0K-000 ref for Customer TSTPRINCE
ABC-0K-000 ref for Customer TSTPRINCE
XLA-Q9-287 ref for Customer MARKSBROS
LLM-8J-665 ref for Customer MARKSBROS
LLM-8J-665 ref for Customer MARKSBROS
LOL-8A-768 ref for Customer KitchenAB
LLM-8J-679 ref for Customer ComputerS
LLM-8J-679 ref for Customer ComputerS
LLM-8J-679 ref for Customer ComputerS
TSW-3P-827 ref for Customer TSTPRINCE
ASD-Q9-425 ref for Customer 671405000
ASD-Q9-425 ref for Customer 671405000
BRA-8J-150 ref for Customer 101564250
GAT-Q9-000 ref for Customer SupliesAB
CEP-8J-552 ref for Customer 020505048
CEP-8J-552 ref for Customer 020505048
>>>> END OF FILE
      23 ORDERS GENERATED

<------------ END OF PROG1 ------------>
```

Figure A-27 JOB results - PROG1 output

The next step in the JCL output is the execution of WebSphere Transformation Extender. To do this, type an S on the next line (the next STEP in the execution output), which is the line that starts with DTXLOG, and press the right Ctrl key to send to view the WTX log. (See the next line in Figure A-26 on page 702.)

Figure A-28 shows a view of the WebSphere Transformation Extender log. The version and build number of WebSphere Transformation Extender are shown. The command line is then displayed as interpreted by WebSphere Transformation Extender for the first map. For each map, card details are displayed, with the default name to use for DDNAME.

```
 SDSF OUTPUT DISPLAY WTXSCEN3 JOB15433  DSID   113 LINE 0       COLUMNS 02- 81
 COMMAND INPUT ===> _                                          SCROLL ===> PAG
********************************* TOP OF DATA ***********************************
*** IBM Websphere Transformation Extender Execution Log for Map MYMAP1 ***
Batch Command Server 8.2.0.3(58)

MYMAP1 /TS /Z28 /I1 TXIN /VX15 TXIN /I2 XREF /VX15 XREF /O1 OPROG2 /VX15 OPROG
 /O2 OPROG3 /VX15 OPROG3 /O3 OPROGXML /O4 ORDERR /VX15 ORDERR

Map MYMAP1 has 6 cards (compilation level 7)
Input card 1
  - Card name        : LegacyFile
  - Source name      : LegacyError.txt
  - Source type      : Data file
  - Processing Mode  : Integral
  Default is ddname LEGACYER
Input card 2
  - Card name        : XrefFile
  - Source name      : Xref.txt
  - Source type      : Data file
  - Processing Mode  : Integral
  Default is ddname XREF
Output card 1
```

Figure A-28 JOB results - DTXLOG file

Execution options are then displayed as shown in Figure A-29.

```
 SDSF OUTPUT DISPLAY WTXSCEN3 JOB15433  DSID   113 LINE 44       COLUMNS 02- 81
 COMMAND INPUT ===> _                                          SCROLL ===> PAG

Execution options:
Input  1 override: DD:TXIN; append; rollback
DD:TXIN will use record separator(s) x15
Input  2 override: DD:XREF; append; rollback
DD:XREF will use record separator(s) x15
Output 1 override: DD:OPROG2; rollback
DD:OPROG2 will use record separator(s) x15
Output 2 override: DD:OPROG3; rollback
DD:OPROG3 will use record separator(s) x15
Output 3 override: DD:OPROGXML; rollback
Output 4 override: DD:ORDERR; rollback
DD:ORDERR will use record separator(s) x15
Memory page size 65536 bytes, page count 8
Created workfiles are on disk
No input trace requested
No output trace requested
Summary trace requested
Validate all input cards
Validate object restrictions
Validate object size
Validate object presentation
```

Figure A-29 JOB results - DTXLOG file

The execution log shows input and temporary files used (Figure A-30). The number of objects that are found and built are displayed.

```
 SDSF OUTPUT DISPLAY WTXSCEN3 JOB15433  DSID   113 LINE 66       COLUMNS 02- 81
 COMMAND INPUT ===> _                                          SCROLL ===> PAG
Audit report not requested
Start year for YY date format without CC 1950
Ignore mapping return code(s) 28

Begin map execution:
Create work file
Dev DISK, org SEQ, recfm FBS blksize 32760 lrecl 32760

Open TXIN
Dev DISK, org SEQ, recfm VB blksize 27998 lrecl 143

Create input work file 01
Dev DISK, org SEQ, recfm FBS blksize 32760 lrecl 32760

Open XREF
Dev DISK, org SEQ, recfm VB blksize 6233 lrecl 129

Create input work file 02
Dev DISK, org SEQ, recfm FBS blksize 32760 lrecl 32760

Create OPROG2
Dev DISK, org SEQ, recfm V blksize 151 lrecl 143
```

Figure A-30 JOB results - DTXLOG file

The map execution report is given with the return code and CPU time (Figure A-31). The last information is about the use of temporary files. These lines are important to check how many files are used. The log deals with the next map the same way.

```
 SDSF OUTPUT DISPLAY WTXSCEN3 JOB15433  DSID   113 LINE 100      COLUMNS 02- 81
 COMMAND INPUT ===> _                                          SCROLL ===> CSR

Burst 1 : elapsed CPU time 0.06 seconds
  Objects found in burst: 286
  Objects built in burst: 243
  Total objects found: 286
  Total objects built: 243

Mapping return code 0, severity NONE:
Map completed (0): Input type contains errors
Job step return code is 0
Elapsed CPU time 0.08 seconds
Virtual storage used by engine 3858089 bytes

File Usage:
Map file, sysname MYMAP1, dsn WTX.SCEN3.MAPS(ORD2NEW)
  Dev DISK, org PDSMEM, recfm FB blksize 32720 lrecl 80
Work file, sysname SYSTMP01, dsn SYS08269.T151147.RA000.WTXSCEN3.TEMP01.H01
  Dev DISK, org SEQ, recfm FBS blksize 32760 lrecl 32760
Input 1:, sysname TXIN, dsn WTX.DATA.ORDERS
  Dev DISK, org SEQ, recfm VB blksize 27998 lrecl 143
Copy of input 1, sysname SYSTMP02, dsn SYS08269.T151147.RA000.WTXSCEN3.TEMP02.
  Dev DISK, org SEQ, recfm FBS blksize 32760 lrecl 32760 size 0
```

Figure A-31 JOB results - DTXLOG file

9. Before we look at the next step of JCL, look at the data files that are generated by WebSphere Transformation Extender with the first map MYMAP1 - ORDS2NEW - OrdersToNew:

   a. To return to the first session, press the F9 key.
   b. Press the F3 key to return to the WTX list of data sets.
   c. In the command line, type `REFRESH` to view the new generated data files.
   d. In Figure A-32 on page 706, type a `B` (for browse) in front of the `WTX.DATA.ORDERS2` file and press the right Ctrl key to send the command.

```
  Menu  Options  View  Utilities  Compilers  Help
 ------------------------------------------------------------------------------
 DSLIST - Data Sets Matching WTX                          Row 1 of 17
 Command ===> _____________________________________________ Scroll ===> PAGE

 Command - Enter "/" to select action                Message          Volume
 ------------------------------------------------------------------------------
          WTX                                                         *ALIAS
          WTX.COBOL.JCL                                               OP1TSB
          WTX.COBOL.LOAD                                              OP1TSF
          WTX.COBOL.SRC                                               OP1TSB
          WTX.DATA.MAPERRCD                                           OP1TSD
          WTX.DATA.ORDERR                                             OP1TSA
          WTX.DATA.ORDERS                                             OP1TSC
          WTX.DATA.ORDERSX                                            OP1TSB
    b_    WTX.DATA.ORDERS2                                            OP1TSA
          WTX.DATA.ORDERS3                                            OP1TSC
          WTX.DATA.ORDLOG                                             OP1TSB
          WTX.DATA.TEMPORD                                            OP1TSF
          WTX.DATA.XREF                                               OP1TSF
          WTX.JCL                                                     OP1TSC
          WTX.MAPS                                                    OP1TSD
  F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel
```

Figure A-32 JOB results - outputs

Figure A-33 shows the contents that are displayed. There are four records, with the same format as the initial ORDERS file.

```
  Menu  Utilities  Compilers  Help
 ------------------------------------------------------------------------------
 BROWSE    WTX.DATA.ORDERS2                     Line 00000000 Col 001 080
 Command ===> _____________________________________________ Scroll ===> PAGE
********************************* Top of Data ********************************
ASD-Q9-425671405000042Doe, Joe                 20080424        BD-IIF34-CDE00002
ASD-Q9-425671405000042Doe, Joe                 20080424        BD-IIF33-CDE00010
BRA-8J-150101564250013Doe, Jane                20080501200805020EV-RRC32-BLU00010
CEP-8J-552020505048013Brown, Jack              20080501200805020EV-RRC32-BLU00010
******************************** Bottom of Data ******************************









  F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel
```

Figure A-33 JOB results - output card1t

e. Repeat step d on page 705 for the `WTX.DATA.ORDERS3` file. There are 12 records in the file (Figure A-34).

```
  Menu  Utilities  Compilers  Help
 ------------------------------------------------------------------------
 BROWSE    WTX.DATA.ORDERS3                        Line 00000000 Col 00
 Command ===> _____________________________________________ Scroll ===>
******************************** Top of Data ***************************
A12300204-28-200805-11-2008KJ-MRN41-DLX000012CS000115980..........
T23100404-28-200805-01-2008CP-IBN71-DLX002500EA000000940..........
A12300304-28-200805-10-2008KL-MRN19-BLU000340EA000012470..........
A12300304-28-200805-10-2008HS-DDX24-BRN000012BX000034950..........
A12300304-28-200805-10-2008QA-ABC06-SEC000142CS000987650..........
A12300204-28-200805-28-2008KJ-MRN41-DLX000012CS000115980..........
T23100104-28-200805-28-2008AJ-XIR98-SEQ000100PC000010230..........
T23100104-28-200805-28-2008PO-YWE24-RED000195DZ000099670..........
M62102404-24-200805-24-2008BD-IIF34-GUL000025CS000349120..........
M62101305-01-200805-02-2008EV-RRC32-BLU000100EA000100000..........
M62101305-01-200805-02-2008EV-RRC32-BRN000100EA000090000..........
T23100404-28-200805-01-2008CP-IBN71-DLX002500EA000000940..........
******************************* Bottom of Data *************************



  F1=Help     F2=Split    F3=Exit     F5=Rfind    F7=Up       F8=Down     F9=Swa
 F10=Left    F11=Right   F12=Cancel
```

Figure A-34 JOB results - output card 2

f. Repeat step d on page 705 for the `WTX.DATA.ORDERSX` file, but this time type an `E` before the line instead of `B`.

Keep in mind that the `WTX.DATA.ORDERSX` file is supposed to be XML. Because the encoding is UTF-8, the z/OS editor is waiting for EBCDIC.

g. To read the data, use the **source data** command. In the command line of the panel, type and send the `source ascii` command (Figure A-35).

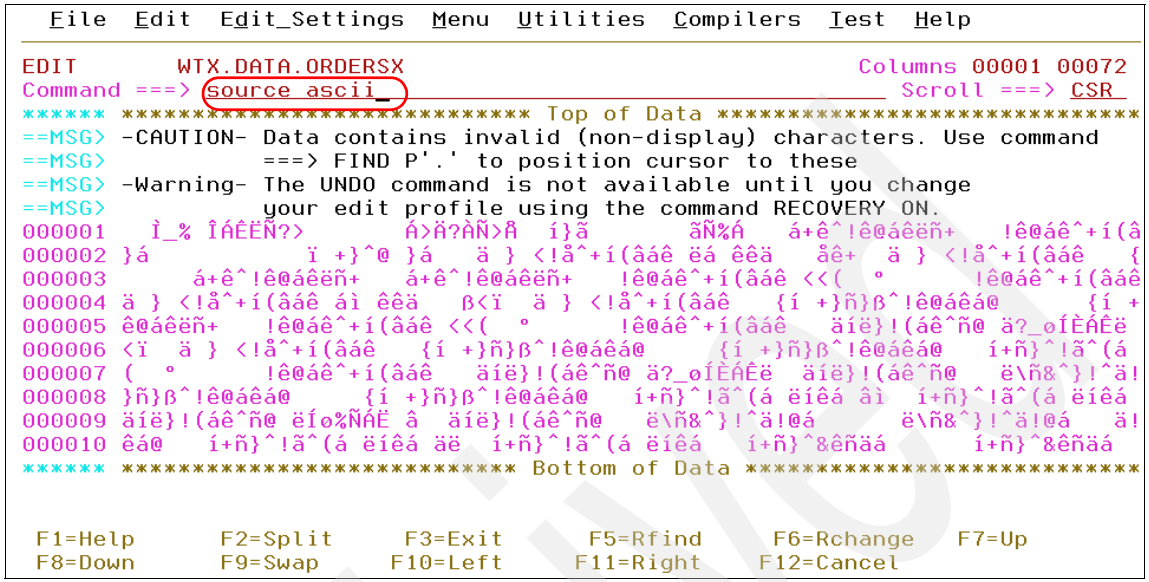

Figure A-35 JOB results - output card 3

Figure A-36 shows how the text should be displayed.

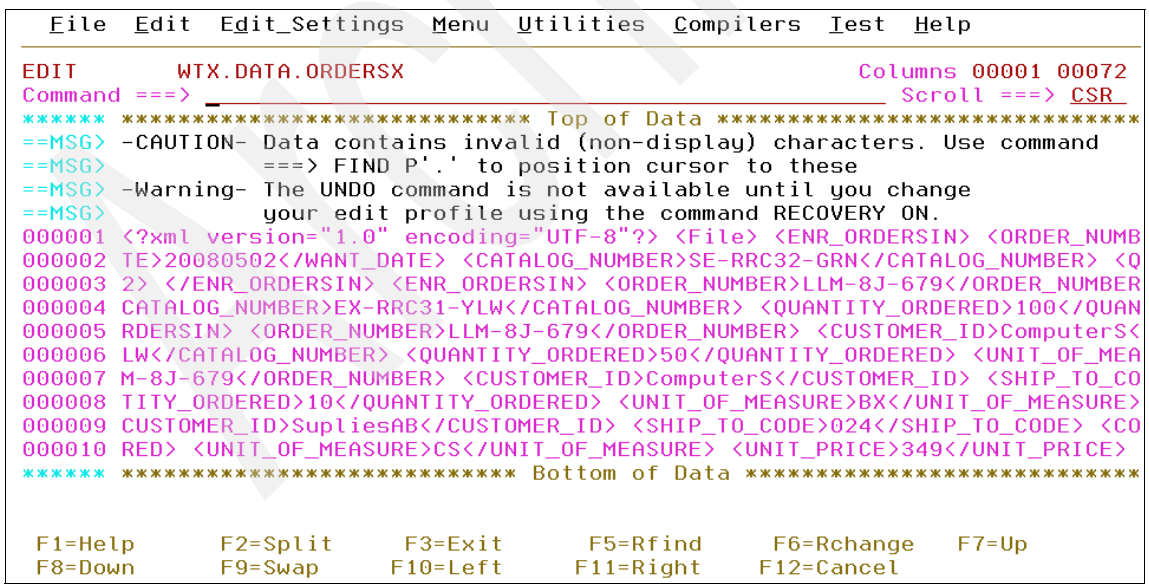

Figure A-36 JOB results - XML output

To read the data in an XML tool, formatted and in colors, download the file with FTP in BINARY mode on a Windows platform. Figure A-37 shows how it should look.

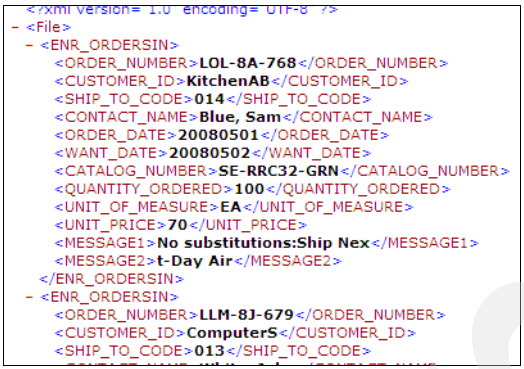

Figure A-37 JOB results - XML output

10. Look at the last file, which is produced with output card 4 of the map. This file contains information about the transformation and is all records with errors. This error file is used by the second map, which is called in the MYMAP2 - ERRDET - ErrorDetails map step. Figure A-38 shows the content of the file.

```
  Menu  Utilities  Compilers  Help
 ------------------------------------------------------------------------------
 BROWSE    WTX.DATA.ORDERR                          Line 00000000 Col 001 080
 Command ===> _____________________________________________ Scroll ===> PAGE
******************************** Top of Data ********************************
Map version  1.02
Input File - DD:TXIN
Process Date:  080925
Time: 15:11:49

The following records are in error:

XRF-1A-150ABCWIDGET003                    2008042820080510KL-MRN19-BLU00034
CEP-8J-552020505048013Brown, Jack         2008050120080502VE-RRC33-BLU00010
****************************** Bottom of Data ******************************




  F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel
```

Figure A-38 JOB results - Error data - Output card 4

The result of MYMAP2 is in the audit log file. This map has no output card. It validates the ORDERR file against the data types and rules in the `OrderError.mtt` type tree.

The DTXAUD audit log has been directed to the `WTX.DATA.ORDLOG` data set. This audit log file shows data type errors and user-defined errors such as the “Want date” compared to the “Order date” (Figure A-39). This audit log file can be used as input by another map that can show clear messages for users. This is not in the scenario, but it data set is typically taught and used.

```
<DataLog>

<input card="1">
 <object index="1" level="1" size="142" status="E07">ENR_ORDERSIN Record</object>
 <object index="1" level="2" size="2"   status="E01">UNIT_OF_MEASURE Field</object>
                                                     <Text>ET</Text>
 <object index="2" level="1" size="142" status="E07">ENR_ORDERSIN Record</object>
 <object index="1" level="2" size="3"   status="E09">SHIP_TO_CODE Field</object>
                                                     <Text>01B</Text>
                                                     <User>Ship-to code must be numeric</User>
 <object index="3" level="1" size="142" status="E07">ENR_ORDERSIN Record</object>
 <object index="1" level="2" size="8"   status="E09">WANT_DATE Field</object>
                                                     <Text>20080502</Text>
                                                     <User>Want date must be after the order

date</User>
 <object index="4" level="1" size="142" status="E07">ENR_ORDERSIN Record</object>
 <object index="1" level="2" size="2"   status="E01">UNIT_OF_MEASURE Field</object>
                                                     <Text>MA</Text>
</input>

</DataLog>
```

Figure A-39 JOB results - Audit log

11. Go back to the other session and press the F9 key to view the results of the other steps.

12. Type an S in front of the line SYSOUT PROG2 to view the PROG2 step. PROG2 reads ORDERS2 file and shows the information in SYSOUT (Figure A-40). The four records are reported.

```
 SDSF OUTPUT DISPLAY WTXSCEN3 JOB15433  DSID   116 LINE 0       COLUMNS 02- 81
 COMMAND INPUT ===> _                                          SCROLL ===> PAGE
********************************* TOP OF DATA ***********************************

-----------------> PROG2 <-----------------

3D-IIF34-CDE ref for Customer 671405000   Amount :      8728.00
3D-IIF33-CDE ref for Customer 671405000   Amount :     10000.00
EV-RRC32-BLU ref for Customer 101564250   Amount :      6990.00
EV-RRC32-BLU ref for Customer 020505048   Amount :     10000.00
>>>> END OF FILE
       4 ORDERS MANAGED
    35718.00 TOTAL ORDERS VALUE

<------------ END OF PROG2 ------------>
******************************** BOTTOM OF DATA *********************************
```

Figure A-40 JOB results - PROG2 output

13. Repeat step 12 to view the results of PROG3 (Figure A-41). In this case, 12 records are reported.

```
 SDSF OUTPUT DISPLAY WTXSCEN3 JOB15433  DSID   118 LINE 0       COLUMNS 02- 81
 COMMAND INPUT ===> _                                          SCROLL ===> PAGE
********************************* TOP OF DATA ***********************************

-----------------> PROG3 <-----------------

KJ-MRN41-DLX ref for Customer A123        Amount :      1437.60
CP-IBN71-DLX ref for Customer T231        Amount :     23500.00
KL-MRN19-BLU ref for Customer A123        Amount :      4998.00
HS-DDX24-BRN ref for Customer A123        Amount :       474.00
QA-ABC06-SEC ref for Customer A123        Amount :    140083.00
KJ-MRN41-DLX ref for Customer A123        Amount :      1437.60
AJ-XIR98-SEQ ref for Customer T231        Amount :      1230.00
PO-YWE24-RED ref for Customer T231        Amount :     18856.50
BD-IIF34-GUL ref for Customer M621        Amount :      8530.00
EV-RRC32-BLU ref for Customer M621        Amount :     10000.00
EV-RRC32-BRN ref for Customer M621        Amount :      9000.00
CP-IBN71-DLX ref for Customer T231        Amount :     23500.00
>>>> END OF FILE
      12 ORDERS MANAGED
   243046.70 TOTAL ORDERS VALUE

<------------ END OF PROG3 ------------>
******************************** BOTTOM OF DATA *********************************
```

Figure A-41 JOB results - PROG3 output

B

Running the test cases for the WebSphere Message Broker scenario

In this appendix, we describe the test cases for the WebSphere Message Broker scenario.

Overview of the test cases

We created nine test cases to test the different scenarios. Depending on the syntax check, the Bank Identifier Code (BIC) check, and the transaction amount, the result of the tests will be different. Table B-1 provides a short overview of the different test cases.

Table B-1 Test cases

Test case	Syntax	BIC return code	Amount
1	NOK	n/a	N/A
2	NOK	n/a	N/A
3	OK	-3	N/A
4	OK	-2	N/A
5	OK	0	< €10000
6	OK	0	< €10000
7	OK	0	> €10000
8	OK	0	> €10000
9	OK	0	> €10000

Each of the test cases is further detailed in the sections that follow.

Test case 1

The test is in the `mt103.1.syntNOK.mbtest` file. In this test, we send a message, which is unrelated to SWIFT, to the input queue. The expected result is for the message to be discovered by the first map and a message to be created on the SCENARIO2_SYNTAX_REJECT queue.

To run test case 1:

1. Double-click the **mt103.1.syntNOK.mbtest** file to see the Unit Test Client (Figure B-1).
2. Click **Send Message** to launch the test.

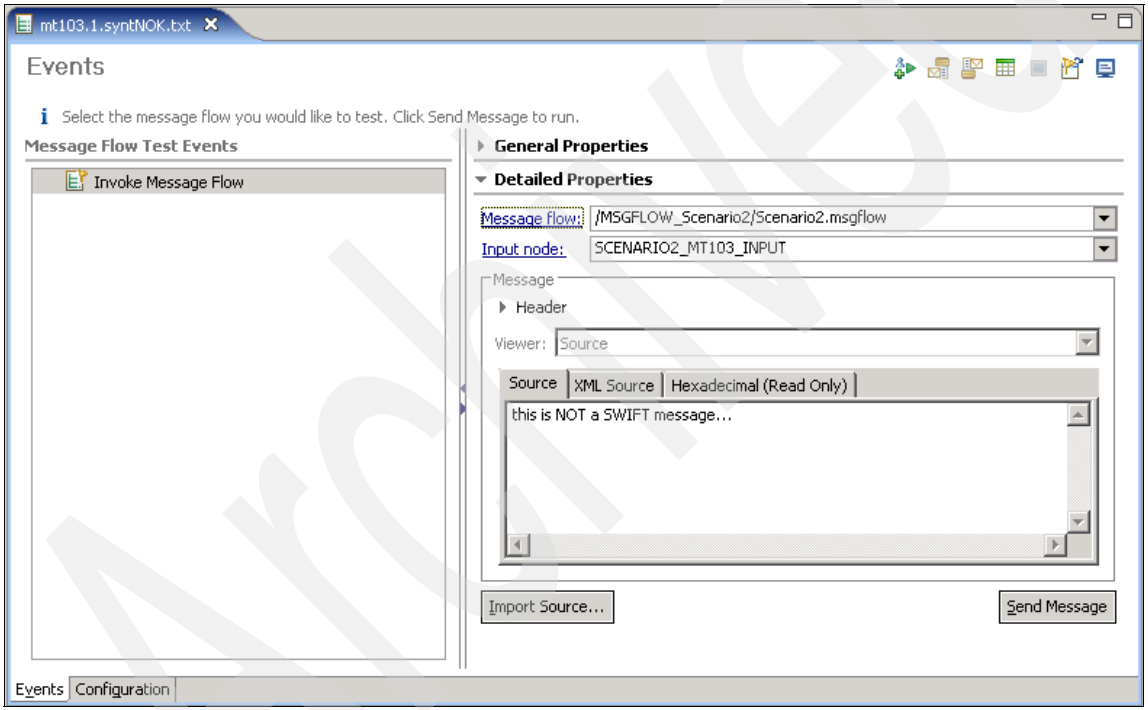

Figure B-1 Test case 1

3. If you see the Deployment Location window (Figure B-2) that prompts you for the deployment location, point to the default execution group and click **Finish**.

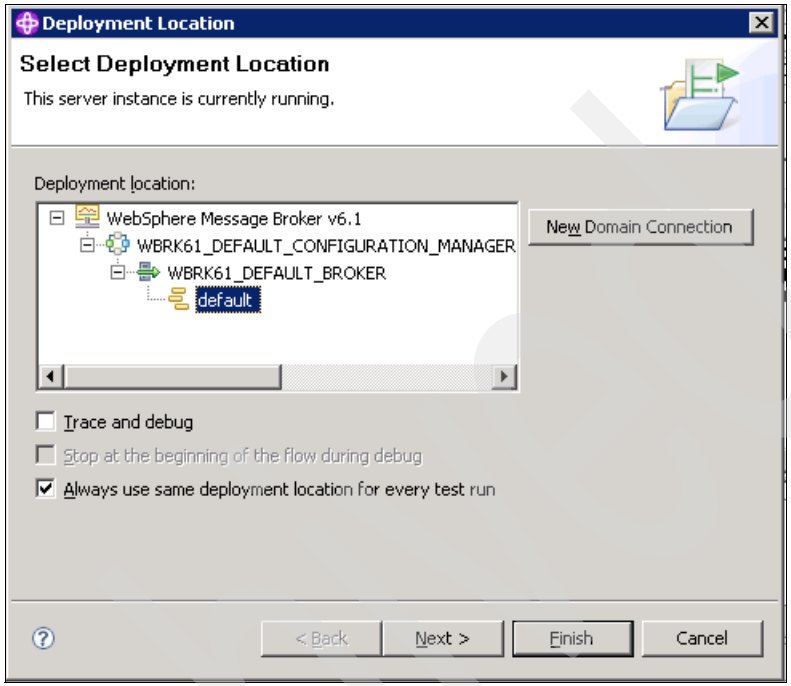

Figure B-2 selecting the deployment location

Figure B-3 shows the result. Notice that the reply message was retrieved from the SCENARIO2_SYNTAX_REJECT queue and that it contains a status field indicating that a syntax error has occurred.

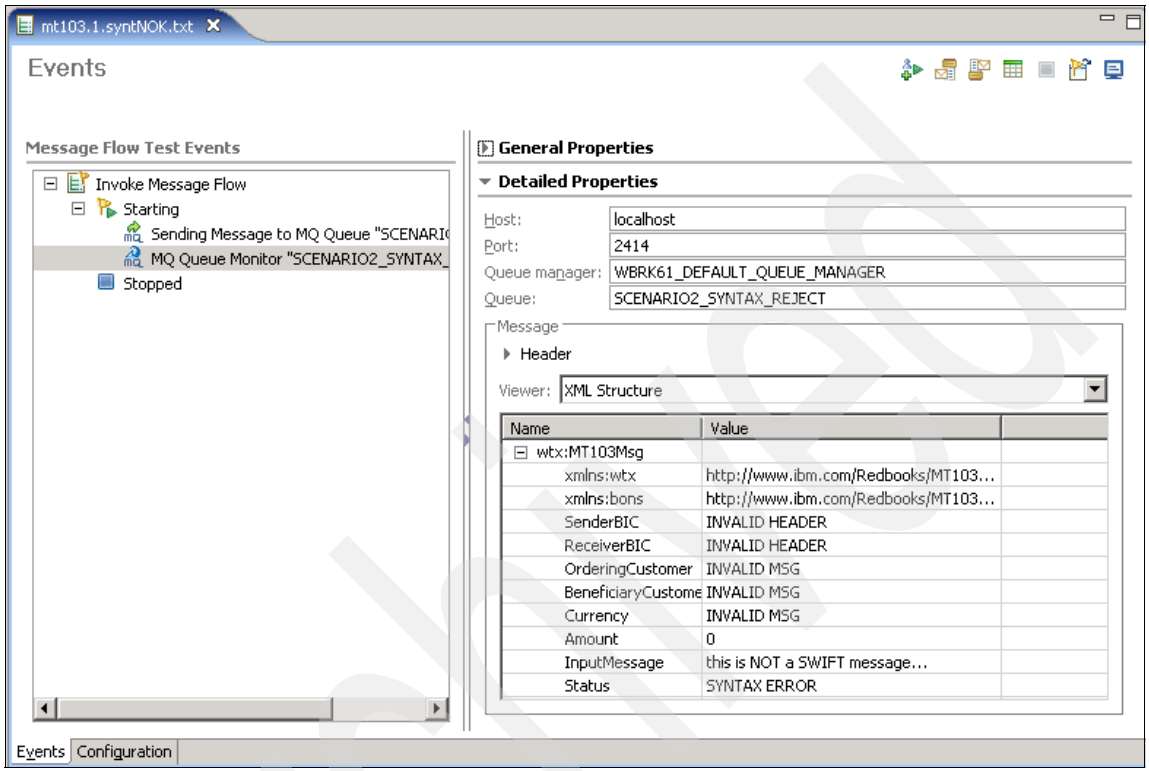

Figure B-3 The result of test case 1

Test case 2

The test is in the `mt103.2.syntNOK.mbtest` file. In this test, we use a SWIFT message, in which we deliberately incorrectly changed one of the amounts. Similar to the previous test, the change qualifies as a bad syntax and a message must be sent to the SCENARIO2_SYNTAX_REJECT queue.

To run test case 2:

1. Double-click the **mt103.2.syntNOK.mbtest** file to open the Unit Test Client for this test (Figure B-4).
2. Click **Send Message** to launch the test.

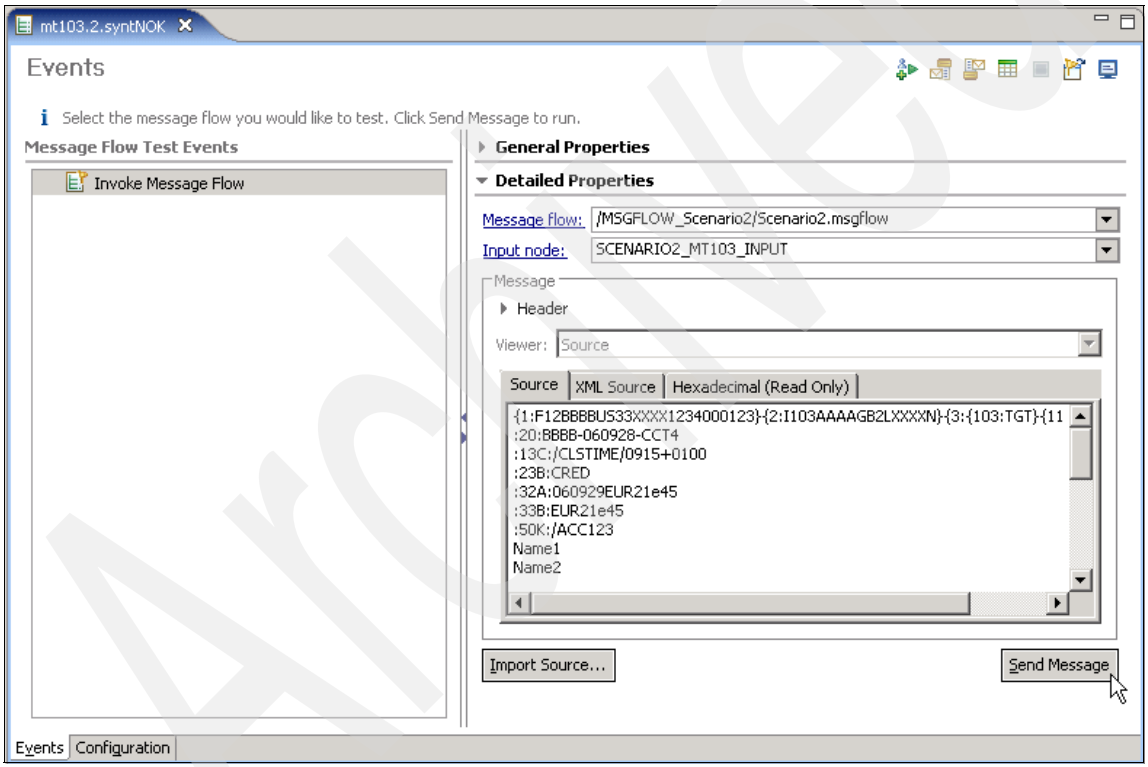

Figure B-4 Test case 2

Figure B-5 shows the result. Notice that the message was retrieved from the SCENARIO2_SYNTAX_REJECT queue and that a syntax error was detected.

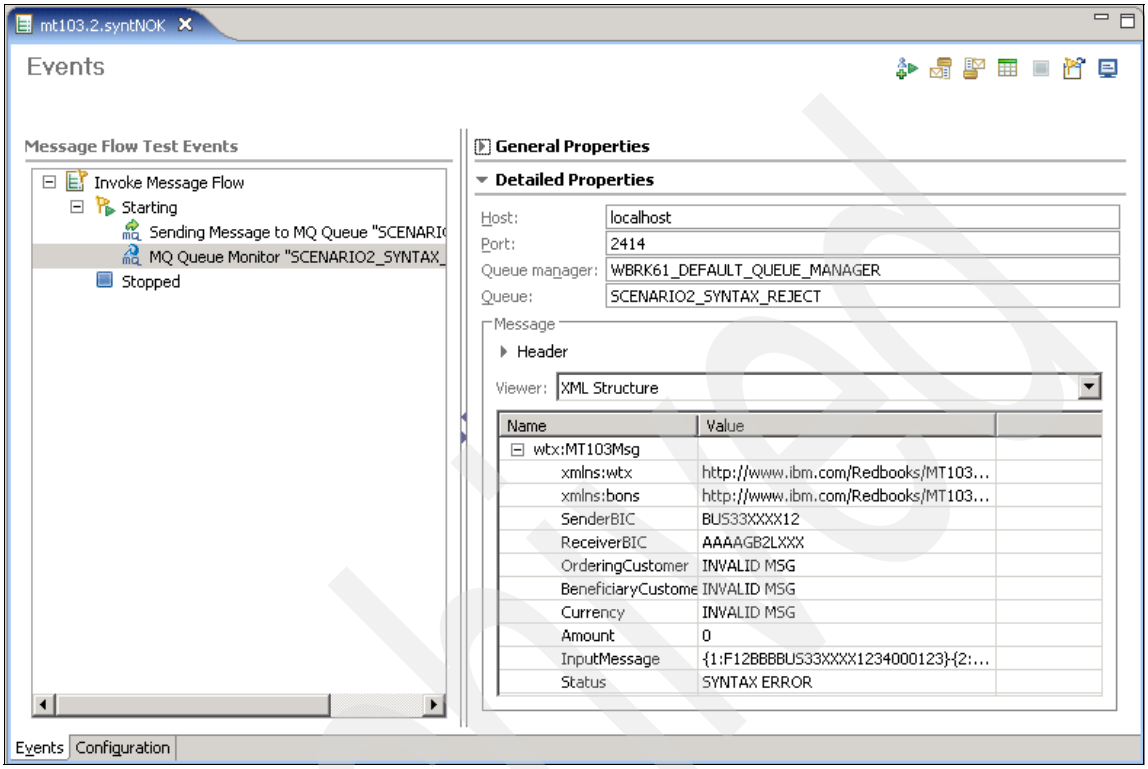

Figure B-5 The result of test case 2

Test case 3

The test is in the `mt103.3.syntOK.bicNoLongerValid.mbtest` file. This test has a syntactically valid SWIFT message, but a BIC that is no longer valid. The expected result of the test is to pass the syntax validation in the first Transformation Extender map and continue its way through the flow. The BIC Web service detects that the BIC is no longer valid. A message must be created on the SCENARIO2_BIC_REJECT queue.

To run test case 3:

1. Double-click the **mt103.3.syntOK.bicNoLongerValid.mbtest** file to open the Unit Test Client (Figure B-6).
2. Click **Send Message** to launch the test.

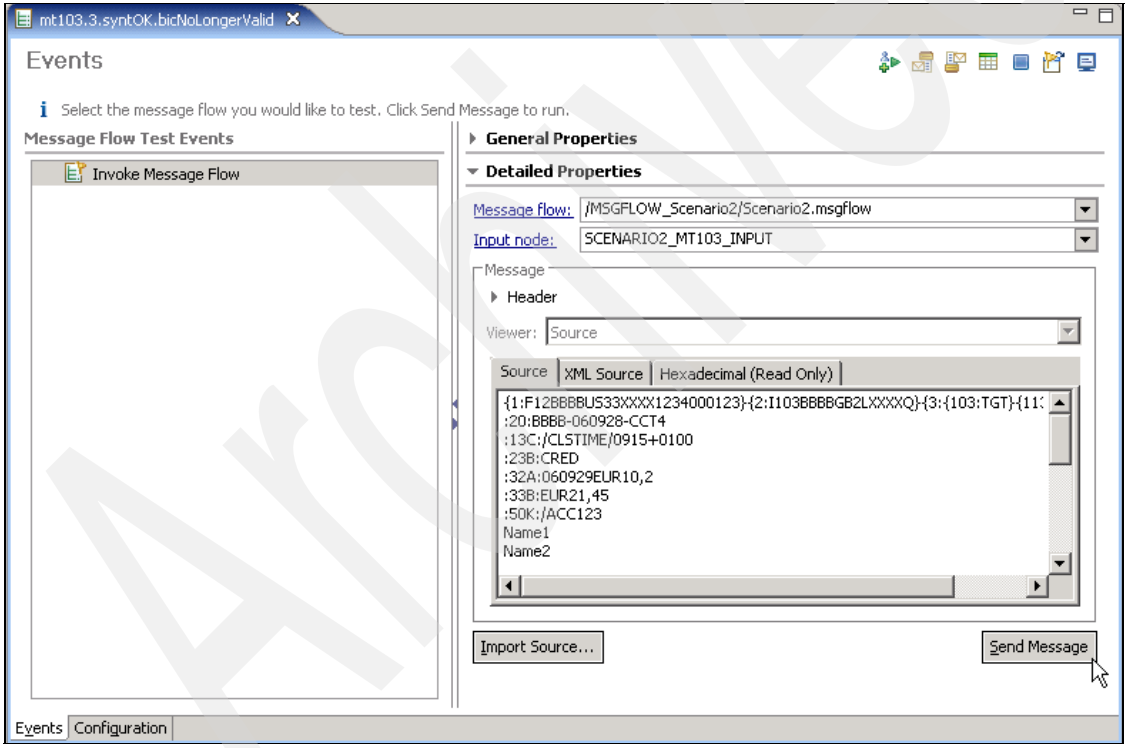

Figure B-6 Test case 3

Figure B-7 shows the result. Notice that the message is retrieved from the SCENARIO2_BIC_REJECT queue. It shows a BIC error -3, meaning that the BIC is no longer valid.

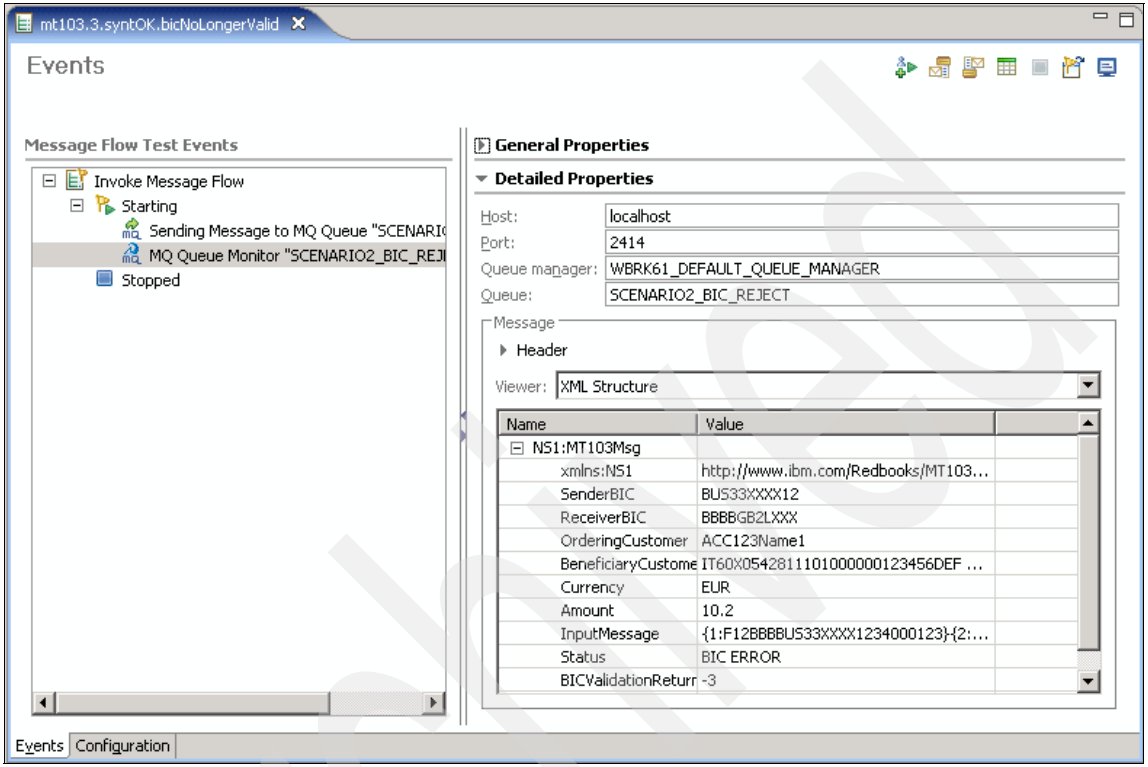

Figure B-7 The result of test case 3

Test case 4

The test is in the `mt103.4.syntOK.bicNotConnected.mbtest` file. In this test, we used a BIC that is invalid. Similar to the previous test, the plan is for this file to trigger the creation of a message on the SCENARIO2_BIC_REJECT queue. However because the BIC is invalid, we should see a return code of -2, meaning that the BIC is not connected to the SWIFT network.

To run test case 4:

1. Double-click the **mt103.4.syntOK.bicNotConnected.mbtest** file to open the Unit Test Client (Figure B-8).
2. Click **Send Message** to launch the test.

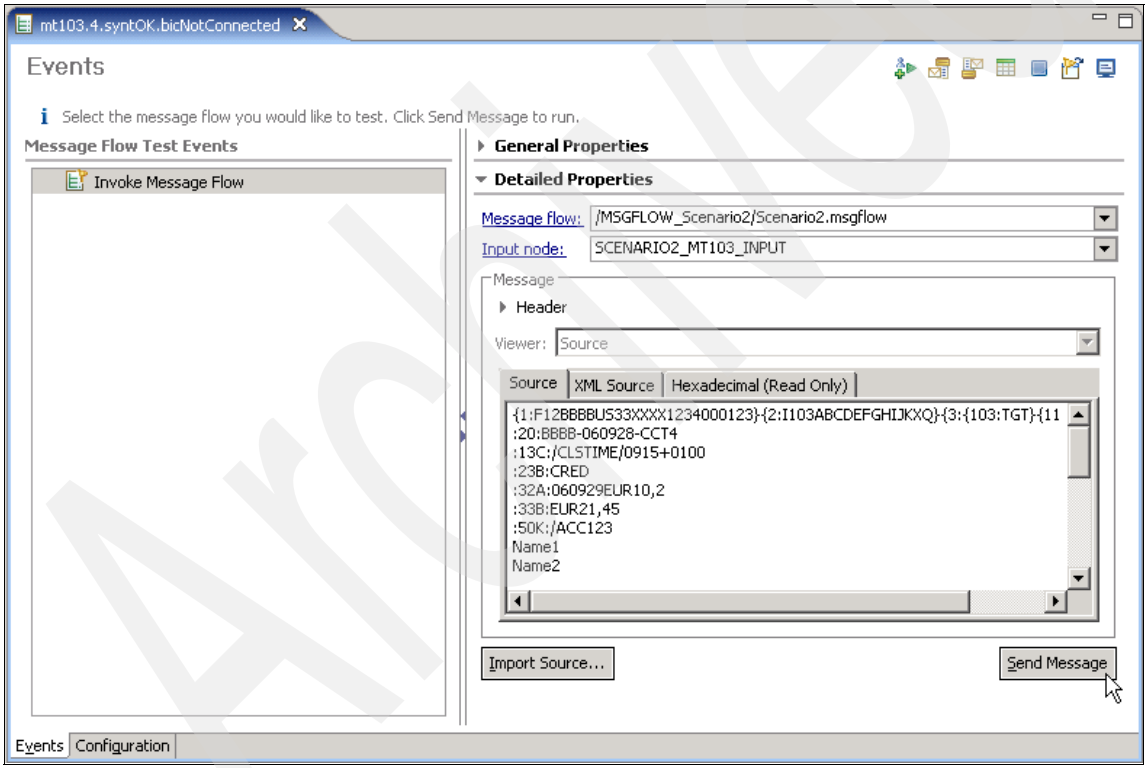

Figure B-8 Test case 4

Figure B-9 shows the result. The message was retrieved from the SCENARIO2_BIC_REJECT queue, and the BIC return code is -2, meaning that the BIC is not connected to the SWIFT network.

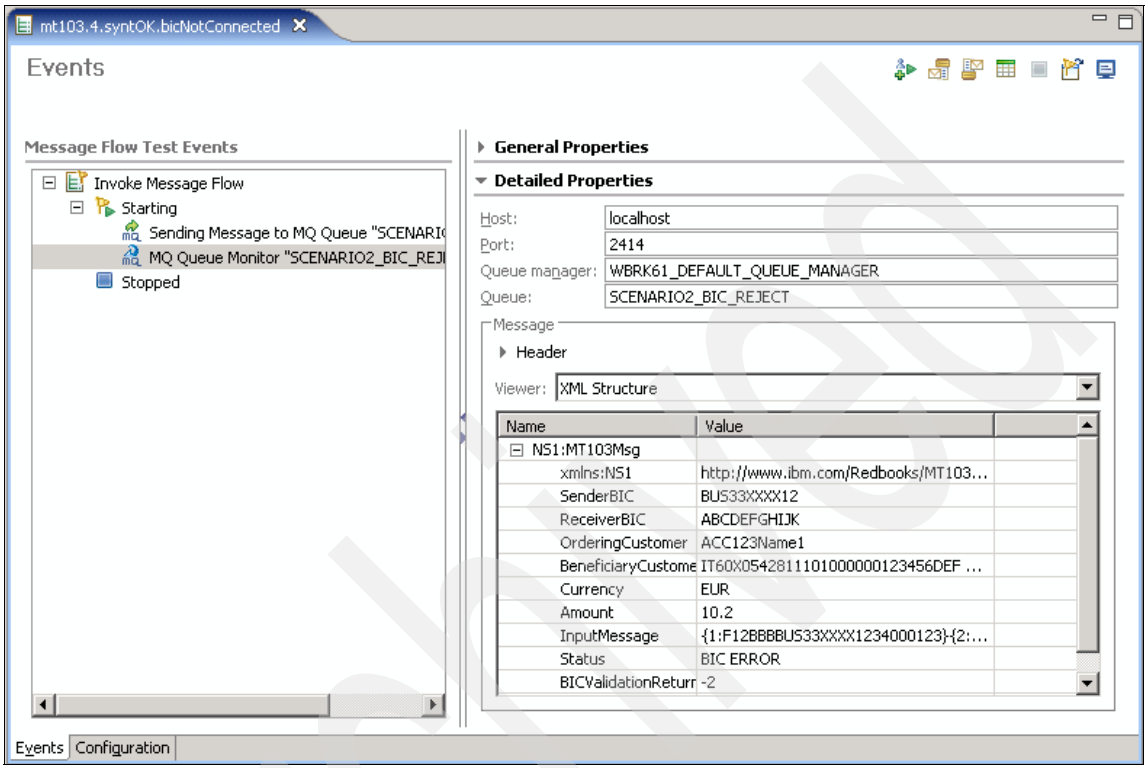

Figure B-9 The result of test case 4

Test case 5

The test is in the `mt103.5.syntOK.bicOK.amtLOW.currEUR.mbtest` file. This test case has a valid syntax and BIC. The amount is expressed in euros and is lower than €10,000. The expected result is that the message will pass the syntax and BIC validation and that the SWIFT message is placed on the SCENARIO2_SWIFT_GATEWAY queue. Because the transaction value is lower than €10,000, no trace file is created.

To run test case 5:

1. Double-click the **mt103.5.syntOK.bicOK.amtLOW.currEUR.mbtest** file to open the Unit Test Client (Figure B-10).
2. Click **Send Message** to launch the test.

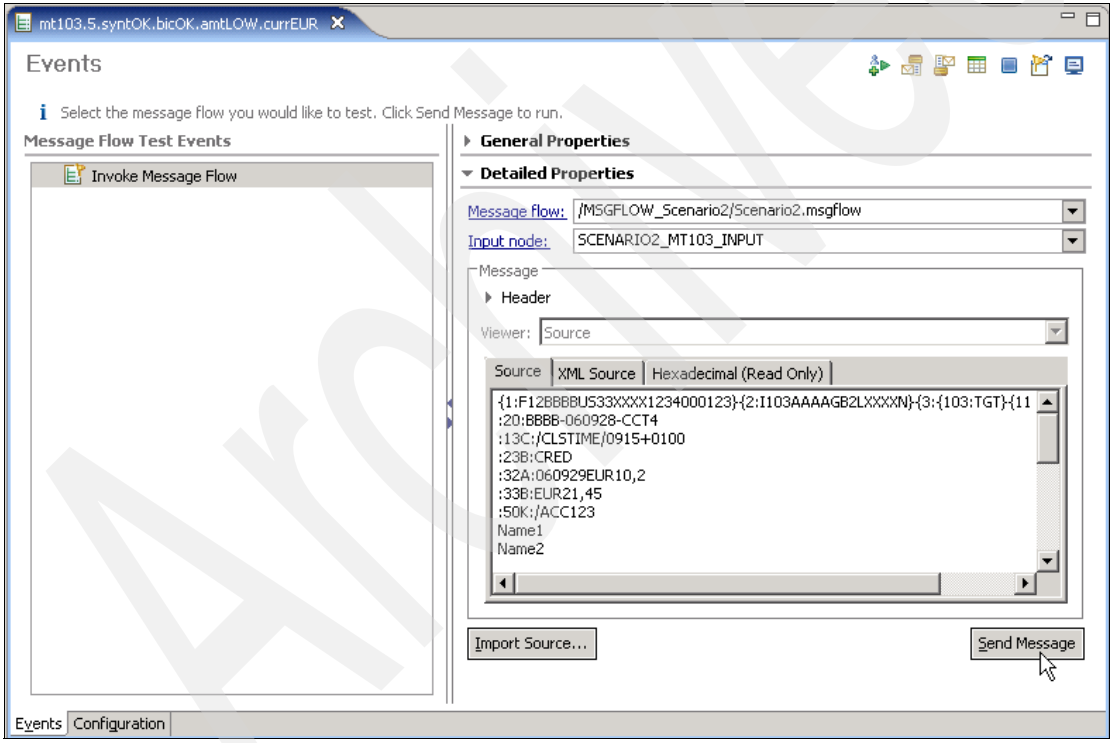

Figure B-10 Test case 5

Figure B-11 shows the result. Notice that the message was retrieved from the SCENARIO2_SWIFT_GATEWAY output queue and passed the validations.

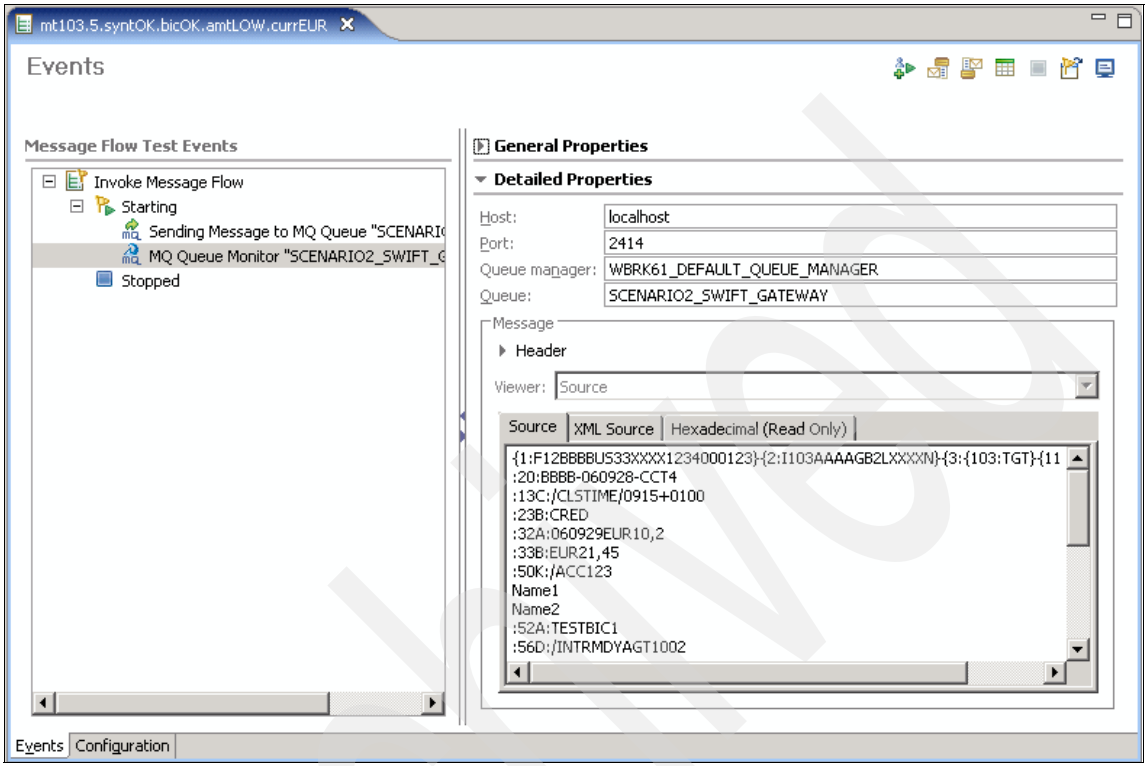

Figure B-11 The result of test 5

Test case 6

The test is in the `mt103.6.syntOK.bicOK.amtLOW.currUSD.mbtest` file. This test is similar to the previous one, but the amount is expressed in U.S. dollars. Therefore, the message must pass through the currency conversion Web service to calculate the transaction value in euros. The expected result is that the message will pass the validations, and the SWIFT message is output on the SCENARIO2_SWIFT_GATEWAY queue. Because the transaction value is lower than €10,000, no trace file is produced.

To run test case 6:

1. Double-click the **mt103.6.syntOK.bicOK.amtLOW.currUSD.mbtest** file to open the test in the Unit Test Client (Figure B-12).
2. Click **Send Message** to send the message to the input queue and start the test.

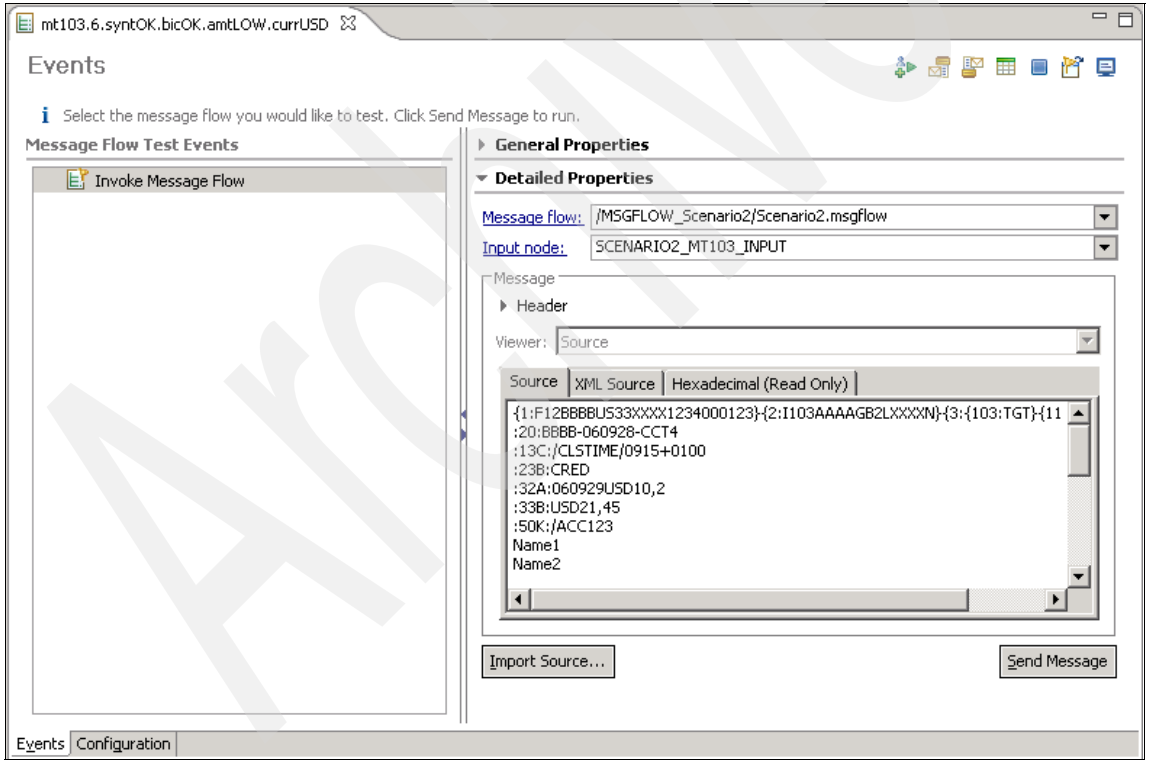

Figure B-12 Test case 6

Figure B-13 shows the result of the test. Notice that the SWIFT message was retrieved from the SCENARIO2_SWIFT_GATEWAY output queue, which means that it passed all validations.

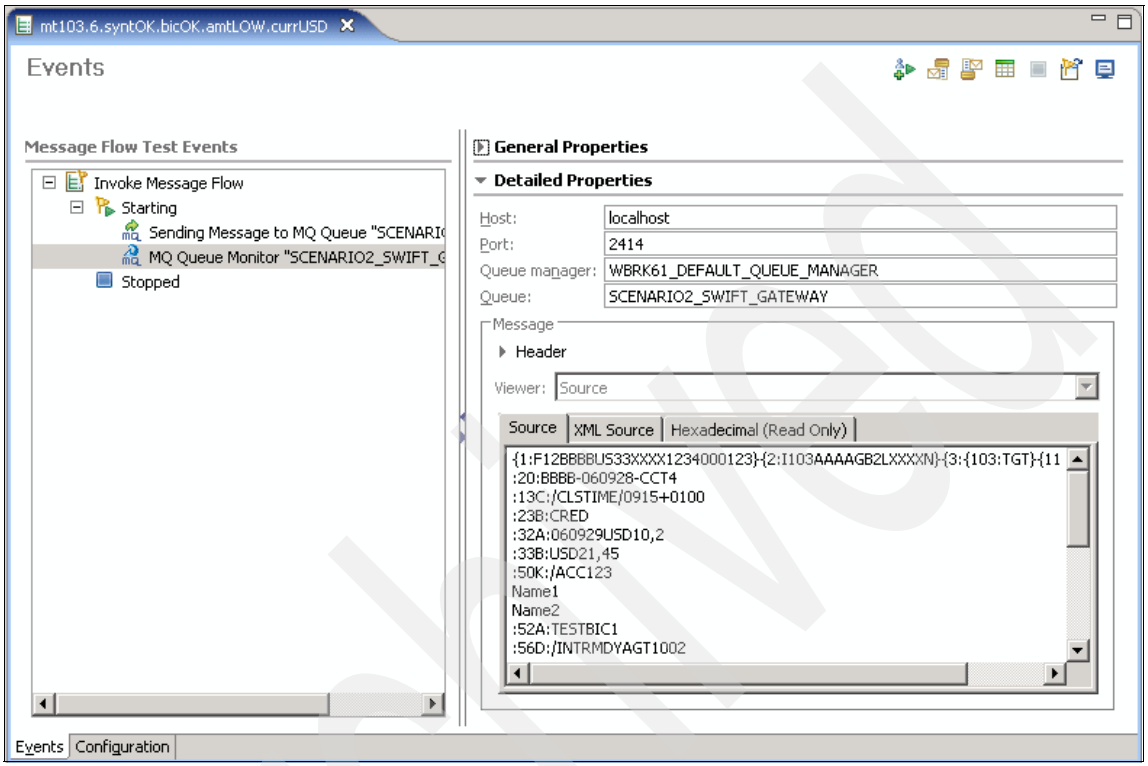

Figure B-13 The result of test 6

Test case 7

The test is in the `mt103.7.syntOK.bicOK.amtHIGH.currUSD.mbtest` file. This file is a SWIFT message that is syntactically correct and has a valid BIC. The transaction amount is expressed in U.S. dollars. Therefore, it must be converted by the currency conversion Web service. The amount in euro is larger than €10,000. Therefore, the Transformation Extender map will initiate the creation of a trace file for this transaction. The expected result is that the original SWIFT message is placed on the SCENARIO2_SWIFT_GATEWAY queue.

To run test case 7:

1. Double-click the **mt103.7.syntOK.bicOK.amtHIGH.currUSD.mbtest** file to open the test in the Unit Test Client (Figure B-14).
2. Click **Send Message** to launch the test.

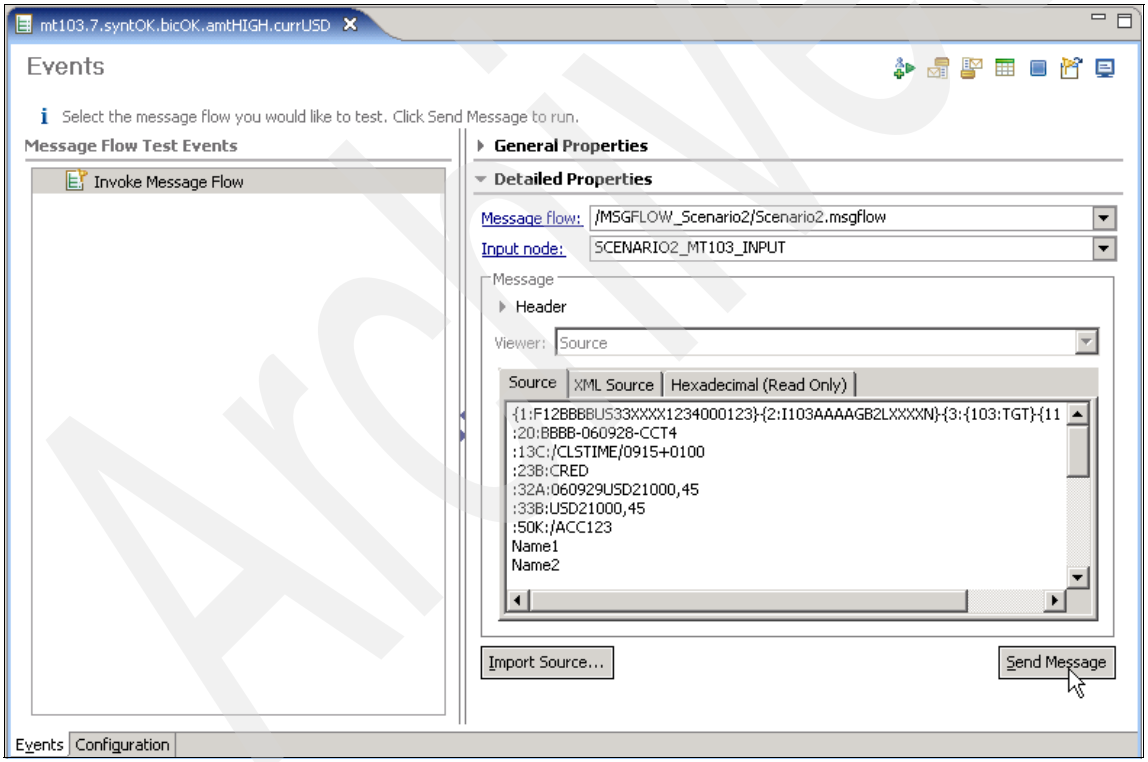

Figure B-14 Test case 7

Figure B-15 shows the result. The SWIFT message was retrieved from the SCENARIO2_SWIFT_MESSAGE output queue, meaning that it passed all validations.

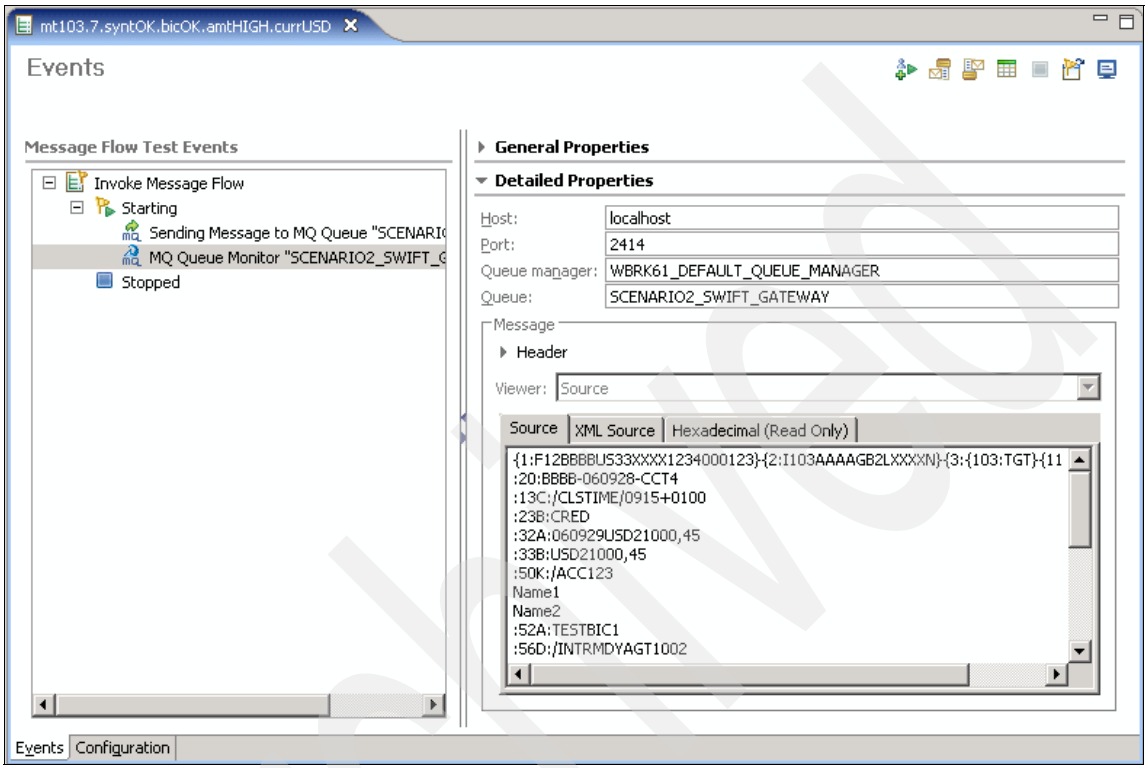

Figure B-15 Test case 7

3. Verify that the trace file was created in the `C:\$Redbook\WMB_Scenario` output directory.

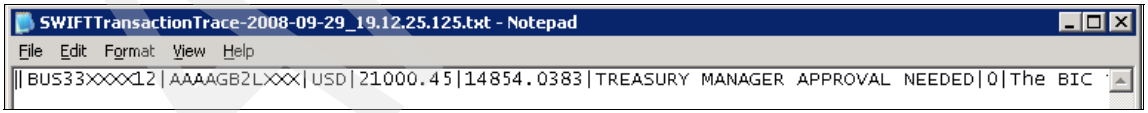

Figure B-16 The trace file for test case 7

Test case 8

The test is in the `mt103.8.syntOK.bicOK.amtHIGH.currEUR.mbtest` file. The transaction amount is expressed in euro. Like the previous test, the expected result is that it should pass all validation steps and create a trace file on output.

To run test case 8:

1. Double-click the **mt103.8.syntOK.bicOK.amtHIGH.currEUR.mbtest** file to open the test in the Unit Test Client (Figure B-17).
2. Click **Send Message** to start the test.

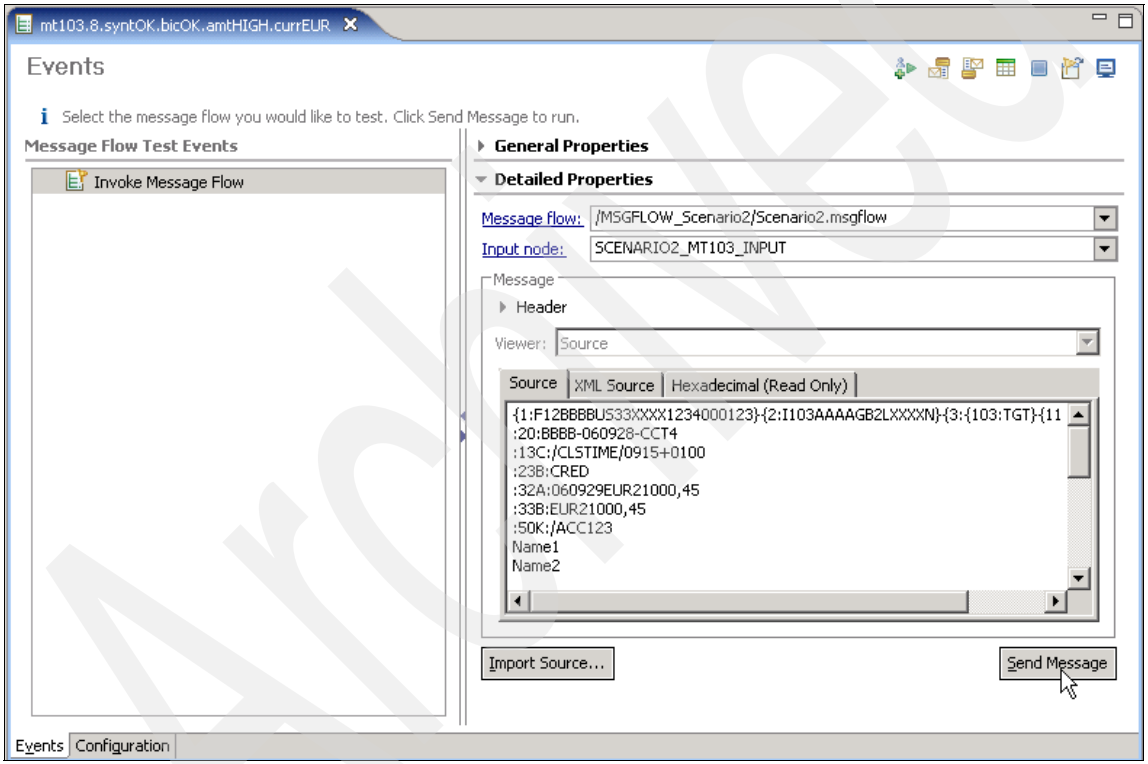

Figure B-17 Test case 8

Figure B-18 shows the resulting message. Notice that the SWIFT message was placed on the SCENARIO2_SWIFT_GATEWAY output queue. Therefore, it passed the validation steps.

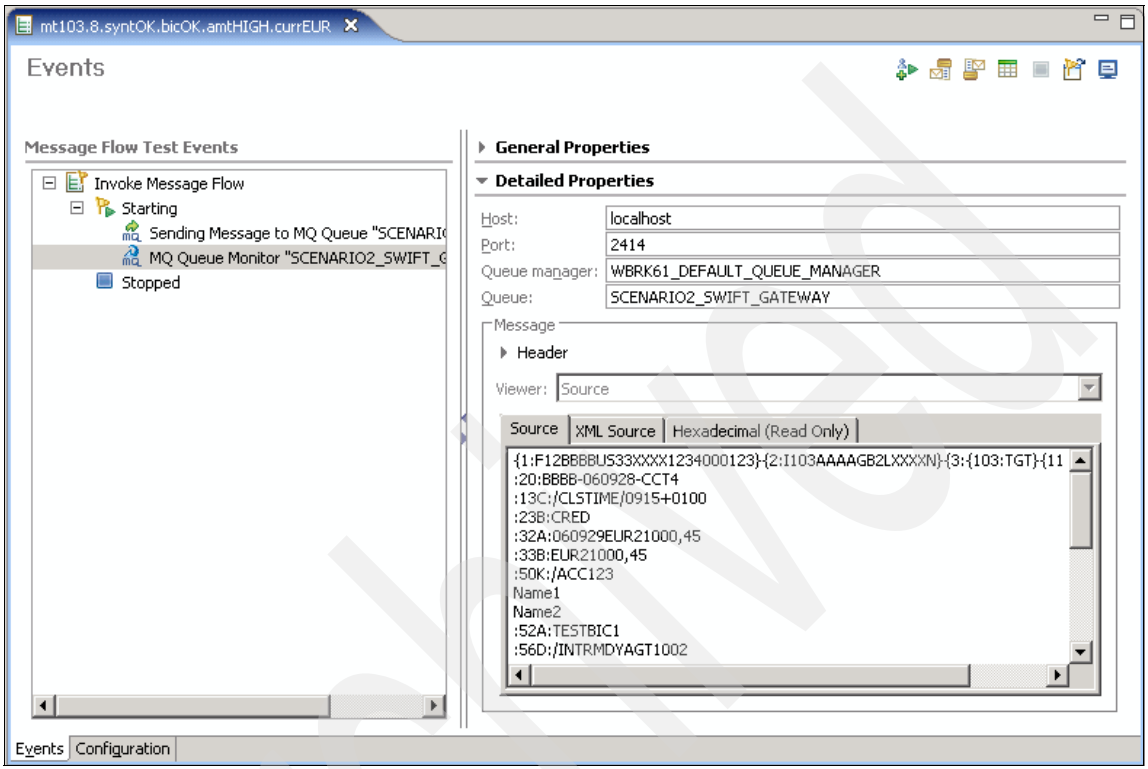

Figure B-18 the result message of test 8

3. Verify that a trace file is created (Figure B-19).

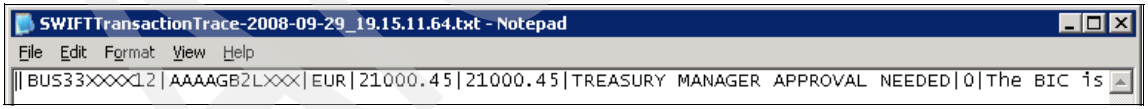

Figure B-19 The trace file for test case 8

Test case 9

The test is in the `mt103.9.syntOK.bicOK.amtHIGH.currGBP.mbtest` file. The transaction amount is expressed in British pounds (£). Like the previous tests, the expected result is that it will pass all validation steps and create a trace file.

To run test case 9:

1. Double-click the **mt103.9.syntOK.bicOK.amtHIGH.currGBP.mbtest** file to open the test in the Unit Test Client (Figure B-20).
2. Click **Send Message** to start the test.

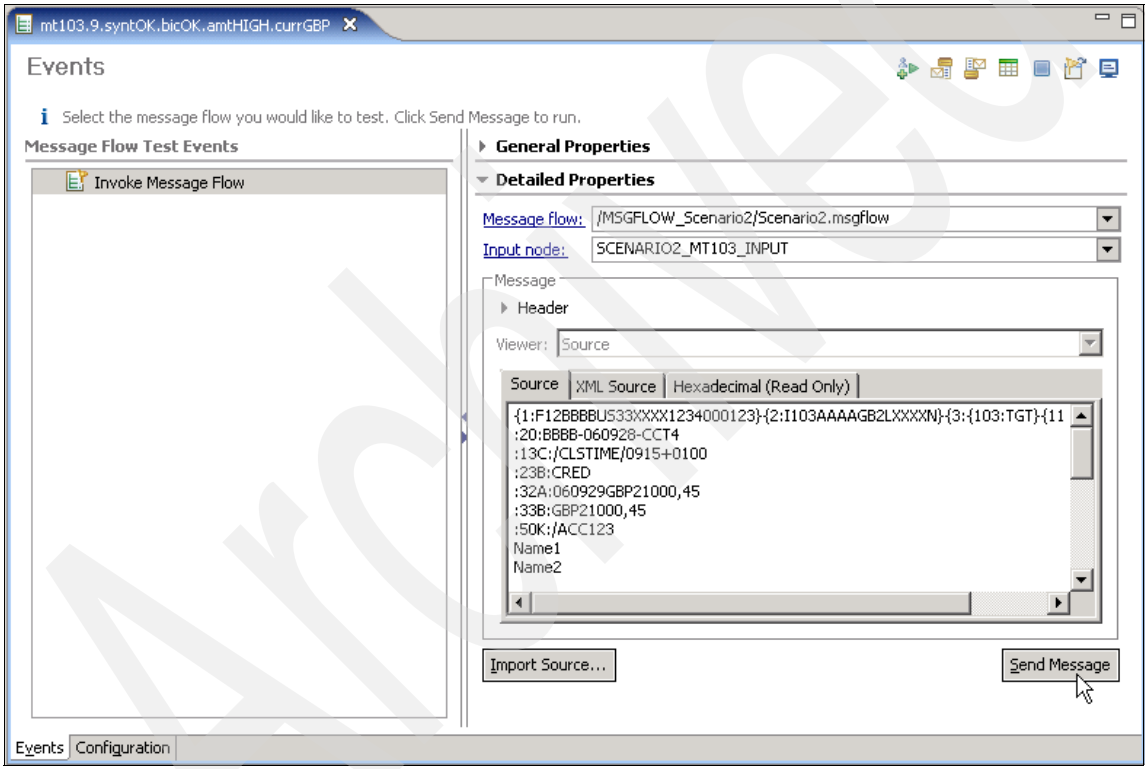

Figure B-20 Test case 9

Figure B-21 shows the resulting message. Notice that the message was retrieved from the SCENARIO2_SWIFT_GATEWAY output queue. Therefore, it passed all validations.

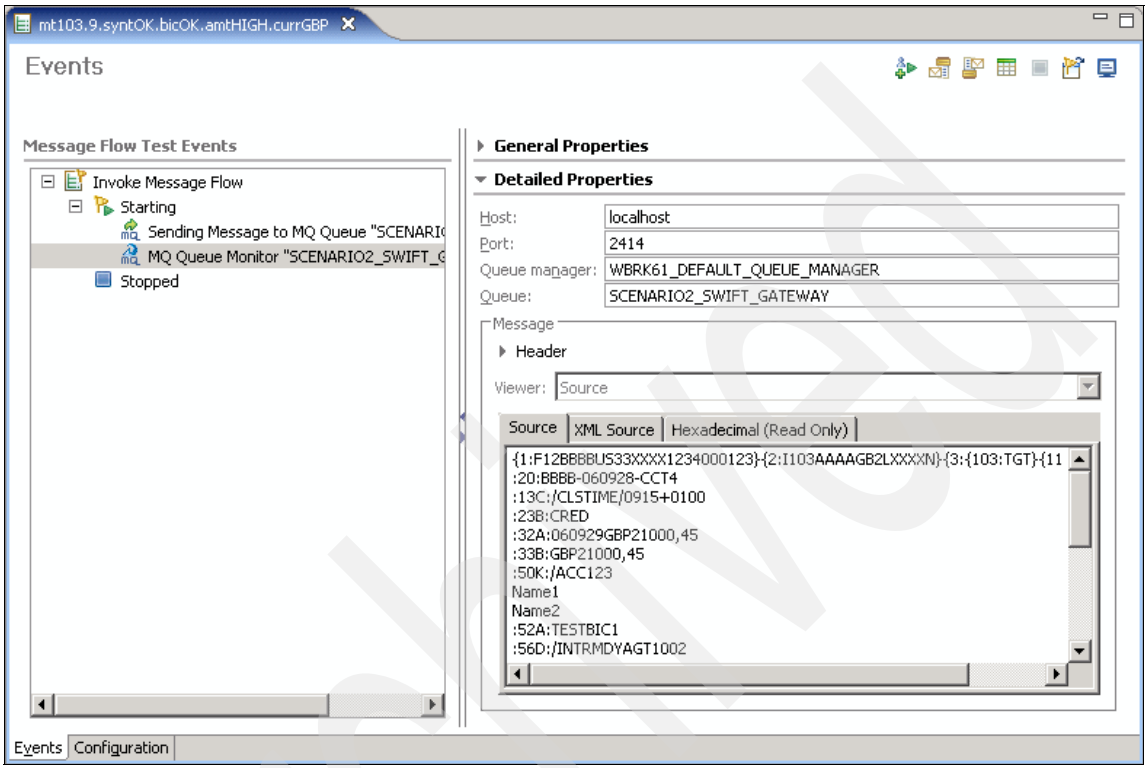

Figure B-21 The resulting message for scenario 9

3. Verify that a trace file is created (Figure B-22).

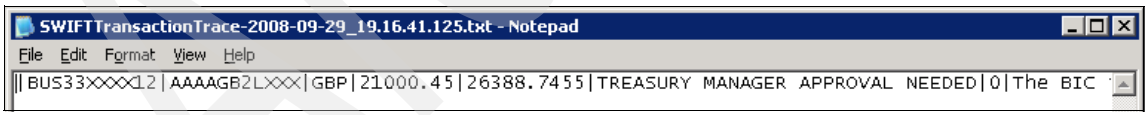

Figure B-22 The trace file for test case 9

C

Using type trees from the SWIFTNet FIN industry pack

In this appendix, we describe the best practice usage of Society for the Worldwide Interbank Financial Telecommunication (SWIFT) type trees within the SWIFTNet FIN industry pack.

Creating a new type tree with the SWIFT generic ISO 7775 type tree from the SWIFTNet FIN Industry Pack

When installed on the development environment, the SWIFTNet FIN industry pack is in the Packs subfolder of the WebSphere Transformation Extender installation path. Inside this Packs subfolder, you can find a folder for each industry pack that is currently installed. One of the folders contains the SWIFTNet FIN pack, called *swift_v,* and the version number (that is *swift_v4.2.3*). The type_trees subfolder contains all type trees of the pack (Figure C-1).

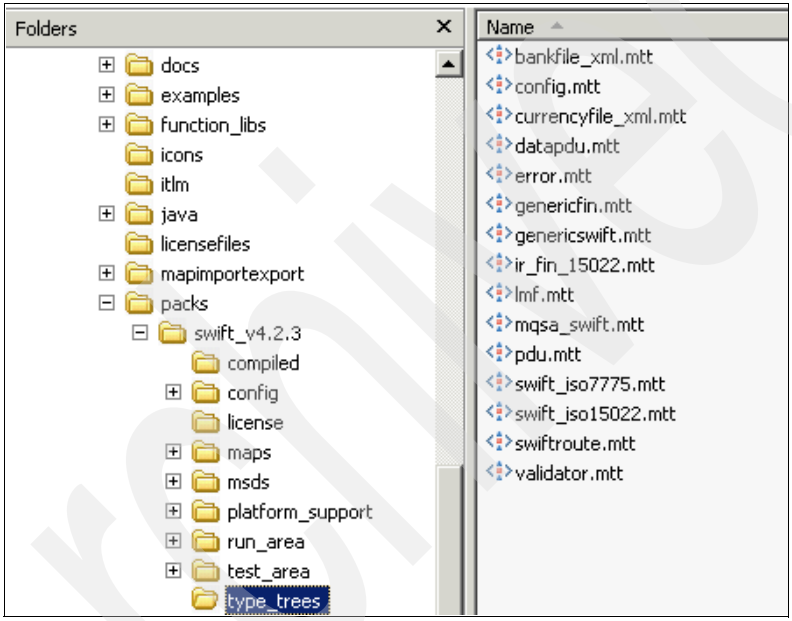

Figure C-1 Installation path of the SWIFT Net FIN industry pack and type_trees subfolder

Two type trees represent all ISO 7775 and 15022 standard message formats:

- The `swift_iso7775.mtt` type tree for ISO 7775
- The `swift_iso15022.mtt` type tree for ISO 15022

More than a simple data structure, SWIFT type trees offer a first step in a validation process by using type properties. They include component rules and restriction lists.

Because the provided trees contain all messages, it is better to generate a customized type tree for the purpose of each specific usage, instead of using this generic type tree as a global format.

To create a new type tree with the SWIFT generic ISO 7775 type tree:

1. Import the generic provided type trees inside your project.

   Even if you do not alter the type trees, *do not* work directly on the type tree files that are provided. We import the necessary files into our project:

   a. Select **File** → **import** → **General** → **File system**.
   b. In the Import – File System window (Figure C-2):

      i. Browse to the **type_trees** subfolder of the SWIFTNet FIN pack installation.

      ii. Select the required type tree or trees.

      iii. For Into folder, browse to the project folder.

      iv. Click **Finish**.

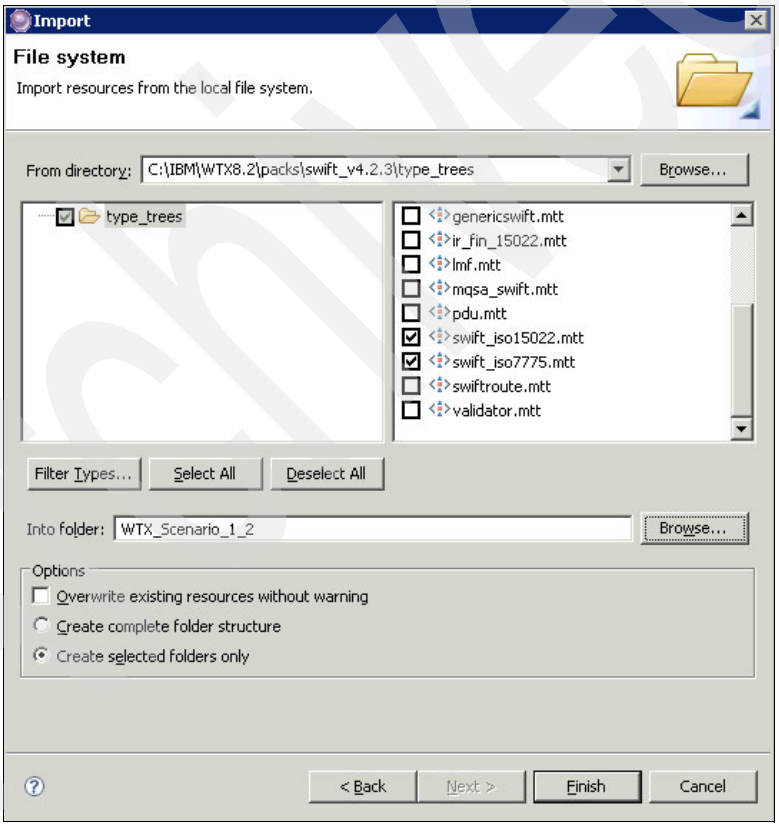

Figure C-2 Importing SWIFT generic type trees into the project folder

   c. Verify that the imported type tree or trees are displayed in the Type Trees folder of the Extender Navigator.

2. Create a new type tree and name the root `SWIFT`. The next step does not work if the root name is different than the SWIFT pack that provided the type trees. Remember that names are case sensitive. Therefore, you must use uppercase.

> **Prefixing SWIFT type trees:** To help you localize type trees if you have many type tree files in your project, prefixing all SWIFT type trees with a logical name, for example `swift_MT103`.

3. In the provided type tree, identify the top-level element that will represent the data stream, depending on the transaction unit of the process. Here, we deal with one message at this time. Therefore, under the Message category, right-click the **MT103 Core** group and select **Merge** (Figure C-3).

> **Note:** Under the Batch category is the top-level element of multiple occurrences of a specific message. Under the Message category, you see the top-level element of one unit of a message.

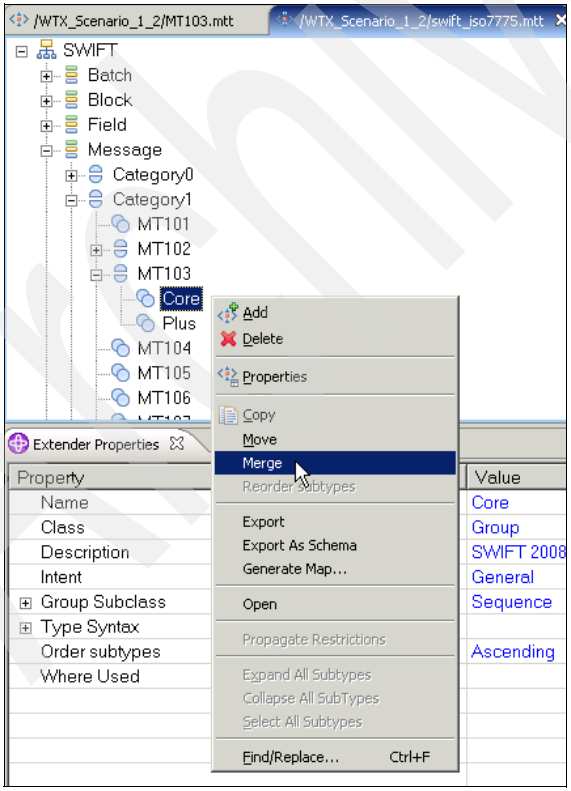

Figure C-3 Merging the Core MT103 group into the customized type tree

4. In the Merge window (Figure C-4), select the new type tree route element (called **SWIFT**) and click **Merge**.

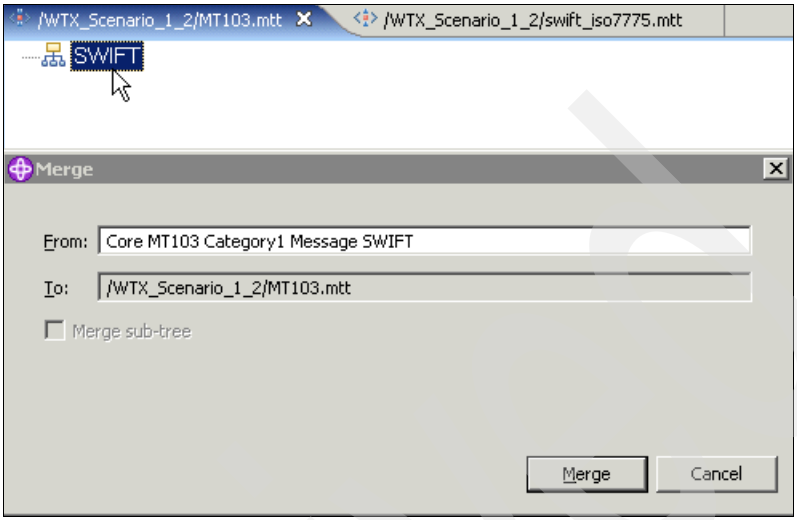

Figure C-4 Selecting the SWIFT root element of the new type tree as destination

5. After the selected Core group *and all its components* are automatically copied to the new type tree, analyze and save the type tree. It is now ready to be used as a validation map.

Using the SWIFT generic ISO 7775 type tree with a template type tree

Many banks and financial institutions use a technical header (Figure C-5 on page 740) that is added to the original SWIFT messages for tracing, processing, monitoring, and routing them into their IT infrastructure. This header is added to any message that is received from a SWIFT computer-based terminal (CBT) and deleted before sending each message to the SWIFT network.

The technical header is held in a template type tree that is used during the merging process. Instead of creating a new type tree, you can open this template, save it under a new name, and then do the merging process.

Sometimes a placeholder (Figure C-5) is set in the template type tree to ensure that the merging process was completed. The placeholder is a component that no longer exists in the type tree to help developers to localize where the merged object should be placed into the template structure. The developers can then replace the placeholder with the merged object.

The fact that this placeholder does not exist as an item of the template type tree generates error message L186 (`Type of component does not exist`) during type tree analysis. (See 5.3.3, “Analyzing type trees” on page 237.) The error message ensures that the replacement was done.

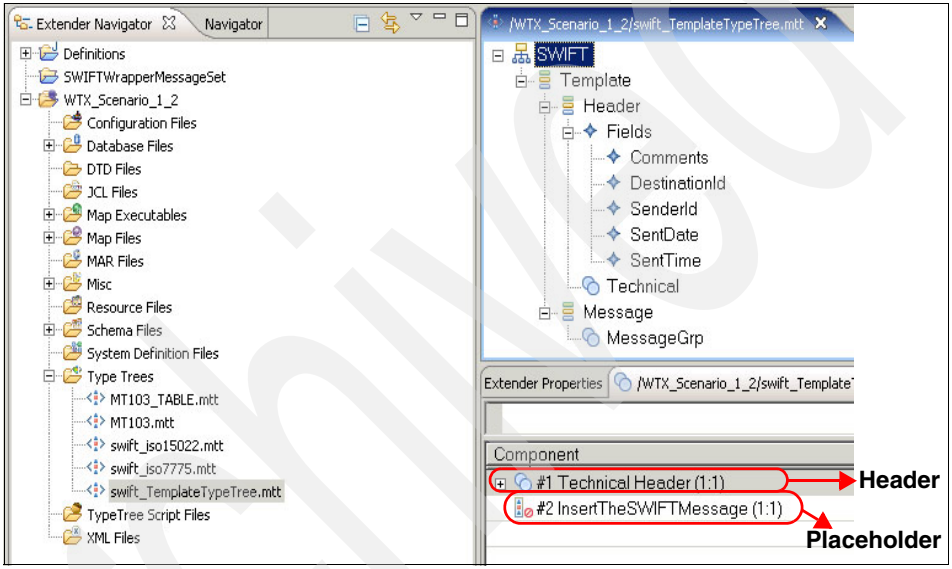

Figure C-5 Template SWIFT type tree with a placeholder and a technical header

D

z/OS SDTXSAMP contents

In this appendix, we describe the contents of the SDTXSAMP partitioned data set (PDS). The contents, listed in Table D-1, include samples for the z/OS environment and data structures that are needed for programming such as the following examples:

- JCL to execute maps in MVS, CICS, and IMS execution environment
- JCL for compiling programs
- JCL for utilities
- Compiled maps
- Data samples
- Data structures for the COBOL, C, and PL/I programs
- Sample programs in COBOL, C, PL/I, and assembler
- Sample programs with DB2 and WebSphere MQ
- Readme files to explain the samples

Table D-1 Listing of members and their description

Member name	Subsystem	Description
DTXALHFS	UNIX System Services	JCL that creates a new hierarchical file system (HFS) for the IBM WebSphere Transformation Extender for z/OS.
DTXALLOC	SMP	Installs.
DTXAMJCL	IMS	Assembles the IMS transaction to DB2 plan cross-reference table.

Member name	Subsystem	Description
DTXASASM	Assembler	Sample program.
DTXASJCL	Assembler	Member that runs the map that drives the assembler exit example (ASMEXIT).
DTXASRME	Assembler	Sample that demonstrates a simple assembler language exit usage with WebSphere Transformation Extender.
DTXB1ASM	IMS	Sample PSB 1.
DTXB1COB	IMS or COBOL	Sample program.
DTXB1JCL	IMS	Example JCL that compiles and links the WebSphere Transformation Extender sample program that is used by the DTXPSB1 transaction.
DTXB1RME	IMS	The readme file that provides information about using the IMS/DC execution option example.
DTXB2COB	IMS	A COBOL program with MQ input and IMS/DC API
DTXB2JCL	IMS	Compiles and Links the DTXPSB2 transaction.
DTXB2RME	IMS	The readme file for the DTXPSB2 transaction.
DTXBIND	DB2	Sample JCL for binding the DBUTIL application plan.
DTXBMJCL	MVS	Sample JCL to perform the reverse map by using BURST MODE processing.
DTXBMRME	MVS	Command Server burst map example readme file.
DTXBMMVS	MAP	Compiled Reverse map example.
DTXBTEXT	Data	Input data for the reverse map.
DTXCCJCL	COBOL	Sample JCL to compile and link the test COBOL API program DTXTCCOB.
DTXCDRME	CICS	Readme DB2 MAP example.
DTXCDMVS	MAP	Compiled (DTXCDMMS) DEPT source map.
DTXDBUTE	DB2	DB2 application plan.
DTXCDMDQ	MDQ	The `dept.mdq` file, the Database Query information for the DEPT (DTXCDMVS) map.
WTXCEJCL	CEE	JCL to create application specific run-time options defaults, CEEWUOPT.
DTXCICPI	COBOL	Definition of the Platform API CardInfo structure.

Member name	Subsystem	Description
DTXCIJCL	CICS	Defines the VSAM KSDS where the WTX maps are stored for use under CICS.
DTXCMRME	CICS	Readme MQSERTST map CICS example.
DTXCMJCL	CICS	Extracts maps and formats them for loading into the VSAM map KSDS.
DTXCQJCL	IMS	Linkedit the WebSphere MQ to IMS dynamic calls stub.
DTXCRDEF	CICS Adapter	Program definitions for the server side of the CICS adapter.
DTXCRMRM	Resource file	The resource file sample.
DTCCRRME	CICS	Readme Resource Registry name example map CICS example.
DTXCTBL	CICS	CICS resource definitions (RDO).
DTXDBUTE	DB2	Bind file.
DTXDDDEF	SMP	Installs.
DTXDHFS	UNIX System Services	The JCL that defines (allocates), mounts, and sets the owner and file permissions for an HFS data set.
DTXDLLSC	C	Sample C language exit that can be called from WebSphere Transformation Extender.
DTXDLLSH	C	Typdedefs for sample.
DTXDMJCL	MVS/DB2	Sample JCL to execute the map to test the DB2 Adapter.
DTXDMMDQ	MVS/DB2	MDQ for DB2 sample.
DTXDMRME	MVS/DB2	Readme for Command Server z/OS DB2 Map Example.
DTXDTCOB	IMS or COBOL	Sample program.
DTXDTJCL	IMS	JCL to compile and link a sample program for IMS/DC.
DTXRXITP	C	ExitParam structure.
DTXEXRME	C, COBOL, or PL/I	Readme file that lists the files for the sample user exits.
DTXTBPSA	COBOL	Source that is used to compile a user exit for WebSphere Transformation Extender.
DTXDLLSC	C	Source used to compile a DLL user exit for WebSphere Transformation Extender.

Member name	Subsystem	Description
DTXPLIXT	PL/I	Source used to compile a user exit WebSphere Transformation Extender.
DTXDLLSH	C	Header file that contains C definitions for the z/OS environment.
DTXEXITP	C	Header file that describes the exit parameter structure.
DTXD2PDS	UNIX System Services	Job that copies a file, created by the Launcher in the UNIX HFS, into a member in a PDS or PDSE data set in the native file system.
DTXDH2SF	UNIX System Services	Job that copies a file, created by the Launcher in the UNIX HFS, into a sequential file in the native file system.
DTXIAEXP	IMS	Imports definitions.
DTXIECOB	COBOL or IMS	Sample INITEP.
DTXIEJCL	COBOL or IMS	JCL that compiles the INITEP subroutine that is called by all of the IMSDC sample programs.
DTXIERME	COBOL or IMS	Readme file that provides information about using the IMS/DC execution option example.
DTXIMJCL	IMS	Execution for IBM WebSphere Transformation Extender MPP.
DTXIMRME	IMS	Readme file that provides information about the modifications that you can make to the IMS/DC system to run the IMS/DC execution option examples.
DTXINST	UNIX System Services	Job that executes the **createuser** shell script in the bin directory of the WebSphere Transformation Extender shared components.
DTXISMKD	SMP	Installation.
DTXMGJCL	MVS or MQ adapter	Sample JCL to test the WebSphere MQ Adapter.
DTXMKDIR	UNIX System Services	REXX to create directories.
DTXMMJCL	DB2	Sample JCL for running MTSMAKER with an MDQ file.
DTXMMASM	IMS	Transformation Extender IMS ATTACH FACILITY TABLE.
DTXMPJCL	MVS or MQ	Sample JCL to test the WebSphere MQ Adapter with the Execution Engine for z/OS (MVS).
DTXMQCOB	IMS	COBOL copybook for WebSphere Transformation Extender WebSphere MQ (formerly WebSphere MQSeries®) and IMS example.

Member name	Subsystem	Description
DTXMQRME	MVS	Readme file that provides information about using the WebSphere MQ map to test the WebSphere MQ adapter with the execution engine for z/OS.
DTXMRM	MVS	Template resource file.
DTXMSJCL	MVS	Sample JCL that demonstrates the Resource Names by using a modified Reverse map.
DTXMSMRM	MVS	Sample resource file.
DTXMSRME	MVS	Readme file that provides information about running the Resource Registry Name example.
DTXMTRME	IMS	Readme file that provides information about using the IMS/DC execution option example.
DTXPGJCL	MVS	Sample JCL that runs the REVERSE map first run in DTXBMJCL with the DTXPAGE utility.
DTXPJCMP	PL/I or MVS	Sample JCL to compile and link the test PL/I API program by using the WebSphere Transformation Extender Platform API for z/OS.
DTXPJRUN	PL/I/MVS	Sample JCL to execute the SINKMAP map by using the WebSphere Transformation Extender Platform API for z/OS PL/I test program.
DTXPL1XT	PL/I or MVS	Program that demonstrates how to call a PL/I program from a map.
DTXPMRME	IMS	Readme file that provides information about using the IMS/DC execution option example.
DTXPRJCL	MVS	DTXPROF utility with the same map used by the DTXBMJCL example.
DTXPROC	UNIX System Services or Launcher	PROC that controls the WebSphere Transformation Extender EVENT SERVER MVS operator console interfaces.
DTXRDRME	IMS	Readme file that provides information about using the IMS/DC execution option example.
DTXRMCOB	COBOL	COBOL definition of the Platform API ExitParam structure.
DTXRRJCL	CICS	Defines the IBM WebSphere Transformation Extender Resource Name VSAM data set.
DTXRTEXT	Data	Input text file for the REVERSE map.
DTXRUNMH	C	Structure definitions (.h).

Member name	Subsystem	Description
DTXRVRME	CICS	Readme file that provides information about running the Reverse Map example.
DTXRVMVS	MAP	Compiled Reverse Map example for CICS.
DTXSCTST	CICS	Sample program of methods to invoke the WebSphere Transformation Extender Command Server on CICS.
DTXSCXIT	COBOL	Sample program exit for WebSphere Transformation Extender.
DTXSMJCL	MVS	Sample JCL to execute the SINKMAP map by using the WebSphere Transformation Extender Command Server for z/OS (OS/390®).
DTXSMRME	MVS	Readme file that provides information about running the Sink Map example to test the API program.
DTXSSRME	CICS	Readme file that lists the sample source files for the CICS COBOL programs.
DTXTAEXP	C, COBOL, or PL/I	Imports definitions (SIDEDECK).
DTXTAJCL	C	Sample JCL to compile the C test program that invokes WebSphere Transformation Extender Platform API for z/OS.
DTXTAPIC	C	Sample program C to run map SINKMAP.
DTXTARME	C	Readme file that provides information about the WebSphere Transformation Extender sample files for the Test API on z/OS example.
DTXTBSA	COBOL	PROGRAM-ID. DTXTPBSP.
DTXTCCOB	COBOL	PROGRAM-ID. 'DTXTESTC'.
DTXTCCPY	COBOL	COBOL definition of the Platform API ExitParam structure.
DTXTCJCL	COBOL	Sample JCL to execute the SINKMAP map by using the WebSphere Transformation Extender Platform API for z/OS COBOL test program DTXTESTC.
DTXTCRME	COBOL	Readme file that provides information about the WebSphere Transformation Extender sample files for the COBOL DLL for the z/OS example.
DTXTMJCL	DB2	Sample JCL for executing TDFMAKER.
DTXTPINC	PL/I	Structure definitions.
DTXTPLI	PL/I	Demonstrates how to call the functions that comprise WebSphere Transformation Extender Platform API from a PL/I program.

Member name	Subsystem	Description
DTXTPRME	PL/I	Readme file that provides information about the WebSphere Transformation Extender sample files for the PL/I DLL for the z/OS example.
DTXTRJCL	MVS	Sample JCL to execute the SINKMAP map by using the WebSphere Transformation Extender Platform API for z/OS DTXTESTA.
DTXWMRME	IMS	Readme file that provides information about using the IMS/DC execution option example.

E

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 to:

ftp://www.redbooks.ibm.com/redbooks/SG247693

Alternatively, you can go to the IBM Redbooks Web site at the following address:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG247693.

Integration scenario 1: WebSphere Transformation Extender for Integration Servers with WebSphere ESB

The additional materials include the `ESB_Scenario.zip` file, which contains the following items:

- `readme.txt`

  This file contains an overview of the compressed file contents.

- `ESB_Scenario_Directories.zip`

  This compressed file creates the correct directory structure, which is expected by the WTX data handler in the WebSphere ESB scenario. Extract this file to the root of your C drive. You see that it builds the `C:\$Redbook\ESB_Scenario` directory. In scenario 2, WebSphere Message Broker writes to this directory. Use any extraction tool to extract the file.

- `ESB_Scenario_Services.zip`

  This Project Interchange file contains the implementation of the Web Service that you use in the WebSphere ESB scenario. To load this file into WebSphere Integration Developer, select **File** → **Import** → **Other** → **Project Interchange** and navigate to the file.

- `ESB_Scenario_Maps.zip`

  This Project Interchange file contains the completed maps for the WebSphere ESB scenario. Import this file in your tools if you do not want to build the maps from scratch, but instead want to integrate them into the flow. To load this file into your tools, select **File** → **Import** → **Other** → **Project Interchange** and navigate to the file.

- `ESB_Scenario_Complete.zip`

  This Project Interchange file contains the completed WebSphere ESB scenario. Load this file if you do not want to build anything, but rather want to see how it works. To load this file into your tools, select **File** → **Import** → **Other** → **Project Interchange** and navigate to the file.

Integration scenario 2: WebSphere Transformation Extender for Integration Servers with WebSphere Message Broker

The additional materials include the `WMB_Scenario.zip` file, which contains the following items:

- `readme.txt`

  This file contains an overview of the compressed file contents.

- `WMB_Scenario_Directories.zip`

  This compressed file creates the correct directory structure, which is expected by the file node in the WebSphere Message Broker scenario. Extract this file to the root of your C drive. You see that it builds the `C:\$Redbook\WMB_Scenario` directory. WebSphere Message Broker writes to this directory. Use any extraction tool to extract the file.

- `WMB_Scenario_Start.zip`

  This Project Interchange file contains the base message flow that you start with. To load this file into your tools, select **File** → **Import** → **Other** → **Project Interchange** and navigate to the file.

- `WMB_Scenario_Maps.zip`

  This Project Interchange file contains the completed maps for the WebSphere Message Broker scenario. Import this file into your tools if you do not want to build the maps from scratch, but rather want to integrate them into the flow. To load this file into your tools, select **File** → **Import** → **Other** → **Project Interchange** and navigate to the file.

- `WMB_Scenario_Complete.zip`

  This Project Interchange file contains the completed WebSphere Message Broker scenario. Load this file if you do not want to build anything but rather want see how it works. To load this file into your tools, select **File** → **Import** → **Other** → **Project Interchange** and navigate to the file.

- `WMB_Scenario_Map_unit_test_files.zip`

  This compressed file contains a set of test data with which you can unit test the maps that you build for the WebSphere Message Broker scenario. When you extract this file, you see that the test files are created in a directory structure, organized by map and input card. You can choose your input test data from these data sets. Use any extraction tool to extract the file.

Depending on the task that you want to achieve, you need the following files:

- To build the maps from scratch and integrate them in the message flow:
  - The `WMB_Scenario_Directories.zip` file to create the output directory
  - The `WMB_Scenario_Start.zip` file (Project Interchange) to import the starting message flow
  - The `WMB_Scenario_Map_unit_test_files.zip` file for unit test data for the maps that you build
- If you do not want to build the maps but instead want to integrate the completed maps into the message flow:
  - The `WMB_Scenario_Directories.zip` file to create the output directory
  - The `WMB_Scenario_Start.zip` file (Project Interchange) to import the started message flow
  - The `WMB_Scenario_Maps.zip` file (Project Interchange) to import the completed maps
- If you do not want to build anything, but instead want to see the solution running:
  - The `WMB_Scenario_Directories.zip` file to create the output directory
  - The `WMB_Scenario_Complete.zip` file to import the complete solution

Integration scenario 3: WebSphere Transformation Extender on System z

The additional materials include the `ZOS_Scenario.zip` file, which contains the following folders. Use any extraction tool to extract the file.

- COBOL
- Copybooks
- Data
- FTPcommands
- JCL
- Maps
- ProjectInterchange

In addition the `ZOS_Scenario.zip` file contains the `Readme.txt` file. This file describes all the contents of these files. One of the files is a Project Interchange file that you can use.

When you import the `WTX_Scenario3.zip` Project Interchange file into Design Studio, you get the complete working scenario. It includes the files that are

necessary to create the scenario from the beginning, as well as the type trees and the maps that we created during the scenario in Chapter 9, “Integration scenario: WebSphere Transformation Extender on System z” on page 623. You can choose one of the following actions:

- Run only the scenario that we created and study the type trees and maps to understand some specific points.
- Create all the artifacts by yourself, as explained in the steps before running the scenario.

Importing the Project Interchange file

To import the Project Interchange file into Design Studio:

1. Open the Design Studio by selecting **Start** → **Programs** → **IBM WebSphere Transformation Extender 8.2** → **Design Studio** → **Design Studio**.
2. Import the Project Interchange file by selecting **File** → **Import**. In the Import window, select the source **Other** → **Project Interchange**. Click **Next**.
3. Browse to the directory of the `WTX_Scenario3.zip` file and select it. Select the project called **WTX_Scenario3** and click **Finish**.

The WTX_Scenario3 project is created with all the artifacts that are needed to run it in Design Studio. To run it in a z/OS environment, check the readme file for this scenario and the steps in 9.2.10, “Running the maps on z/OS” on page 676.

To follow the steps and create the scenario from the beginning, you can create a new project and copy the following files to it:

AllOrders.txt	Contains the test data
Xref.txt	The cross-reference file used for creating the `Xref.mtt` type tree and for running the scenario
Orders.cpy	The COBOL copybook used to create the `Order.mtt` type tree
NewOrders.cpy	The COBOL copybook used to create the `NewOrder.mtt` type tree

These files are in the Misc folder of the imported Extender Project. After copying these files into your new project, follow the steps described in Chapter 9, “Integration scenario: WebSphere Transformation Extender on System z” on page 623, to create the whole z/OS scenario.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

Other publications

The following publications are also relevant as further information sources:

- *IBM XML Toolkit for z/OS Users Guide*, SA22-7932-06

  http://www-03.ibm.com/servers/eserver/zseries/software/xml/pdf/ug_v1r9.pdf

- As described in 1.3, “WebSphere Transformation Extender Online Library” on page 11, a set of Adobe Acrobat PDF files is available that can be installed locally that contains the WebSphere Transformation Extender documentation.

Online resources

These Web sites are relevant as further information sources:

- WebSphere Transformation Extender product page

  http://www-01.ibm.com/software/integration/wdatastagetx/

- WebSphere Transformation Extender support page

  http://www-01.ibm.com/software/integration/wdatastagetx/support/index.html

- WebSphere Transformation Extender Information Center

  http://publib.boulder.ibm.com/infocenter/wtxdoc/v8r2m0/index.jsp

- Various articles about WebSphere Transformation Extender on IBM developerWorks® (including a large collection of articles, tutorials, forums, and other content for developers who use IBM software and technologies)

  http://www.ibm.com/developerworks/search/searchResults.jsp?searchType=1&searchSite=dW&searchScope=dW&query=websphere+transformation+extender&Search=Search

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

IBM WebSphere Transformation Extender 8.2

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

IBM WebSphere Transformation Extender 8.2

Includes key concepts and functionality within WebSphere Transformation Extender

Covers all related editions on distributed and z/OS platforms

Provides integration scenarios

This IBM Redbooks publication provides an introduction to IBM WebSphere Transformation Extender V8.2. WebSphere Transformation Extender is a powerful, transaction-oriented universal data transformation and validation solution. It offers multiple execution options to support right-time, right-style transformation, whether it is batch, real-time, or embedded.

In this Redbooks publication, we present an overview of the WebSphere Transformation Extender products and components and discuss key features for businesses. We describe both the development and runtime or management aspects of the product and review troubleshooting capabilities. This book is written for developers, architects, and IT decision makers who desire a thorough understanding of the various editions and features of WebSphere Transformation Extender.

In this book, we discuss the WebSphere Transformation Extender role in a service-oriented architecture (SOA) working with other components of the IBM Software portfolio, both on distributed platforms and on the IBM System z platform. We include hands-on scenarios that show the end-to-end process to create, deploy, and execute WebSphere Transformation Extender. They show how WebSphere Transformation Extender works with IBM WebSphere Enterprise Service Bus (ESB) and WebSphere Message Broker, as well as on the IBM z/OS environment. This information can help you to understand the product framework as you get started.

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

SG24-7693-00 ISBN 0738432156