z/OS Traditional Application Maintenance and Support

Agile development

Cost containment

Productivity and efficiency

Jonathan Sayles
Chris Rayns
Vijay Sankar
John Milne
Deena Stein
Debasish Dash
Ofer Pharhi

Redbooks

ibm.com/redbooks

International Technical Support Organization

z/OS Traditional Application Maintenance and Support

March 2011

SG24-7868-00

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

First Edition (March 2011)

This edition applies to Version 4, Release 1,CICS Transaction Server, IBM Rational Developer for System z, Version 7, Release 6, Rational Asset Analyzer Version Version 5, Release 5, IBM Debug Tool Version 10, IBM Fault Analyzer Version 10, IBM File Manager Version 10,

Contents

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

Preface

In this IBM® Redbooks® publication, we attempt to provide fresh insight into a problem domain that, in the authors' opinions, has been pushed to the back burner of technology writing for far too long—the domain of IBM z/OS® (traditional) mainframe maintenance and production support. Since the mid-1980's, outside of a few websites and publications, this still-critical area of software has barely even received lip service by the world of mainstream technology media. In a small way, we are attempting address this situation.

In this book, we provide information in "what and how to" sections on the value of z/OS maintenance and support—not the value of the software, which is hardly in question, but the value of the software developers, and how they collaborate, analyze, code, and test the applications, fixes, and enhancements under their responsibility. We present new 21st Century tools to help them achieve their goals more easily and effectively. These tools integrate and provide a 1 + 1 + 1 = 5 value-proposition, for companies that are still doing work the way they did when in the mid-1970's, when Gerald Ford was president of the United States.

We are also describing, to a lesser extent, how you can effectively integrate the new tools with your existing development software stack, in order to find points of complimentary functionality. And we describe the new agile development and maintenance methodologies, and best practices for tools use and adoption.

We hope that you find this work useful, and perhaps that it can fuel more discussion, future Redbooks publications, and other publications by IBM, or any vendor or group interested in this critical and vastly under-acknowledged technology domain.

The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Jonathan Sayles is a z/Platform Technical Evangelist in the U.S. He has 30 years of experience in the IBM System z software field. He holds a Masters degree from the University of Bridgeport. His areas of expertise include IBM DB2/SQL, Workbench Development Tools, COBOL, IBM IMS, and software knowledge transfer. He is published (John Wiley & Sons) in all of these areas.

Chris Rayns is an IT Specialist and Project Leader at the ITSO Poughkeepsie Center in New York. Chris specializes in security and writes extensively about all areas of IBM CICS®. Before joining the ITSO, he worked in IBM Global Services in the United Kingdom as a CICS IT Specialist.

Vijay Sankar is a Senior Application Developer in India. He has five years of experience in the System z® software field. He holds a Masters degree in Computer Application from the University of Calicut. His areas of expertise include IMS™ DB/DC, DB2/SQL, COBOL, and Workbench Development Tools (IBM Rational® Developer for System z, IBM Rational Asset Analyzer, and IBM Rational Team Concert™ for System z).

John Milne is a Systems Programmer with Suncorp Insurance and Banking Group in Australia. He has more than 27 years of experience in a range of mainframe systems areas. He holds a Masters of Information Systems degree from the University of Central Queensland. His areas of expertise include z/OS, DB2®, CICS Transaction Server, IBM WebSphere® MQ, IBM RACF®, application support, and mainframe integration.

Deena Stein is President of Rational Software Solutions Inc., an IBM Business Partner specializing in the Rational and IBM ILOG® product lines. She has 35 years experience in Information Technology and holds an M.B.A. in Computer Methodology from Baruch College of The City University of New York. Her areas of expertise include program/project management, application development, process improvement, methodology, and metrics. She can be reached at dstein@RationalSoftwareSolutions.com.

Debasish Dash is associated with IBM, India for last 6 years. He has 13 years of strong IT experience covering Technical Consultancy, Project / Program Management and Competency Development, mostly on Mainframe and Web technologies. He holds a Bachelors degree in Electrical Engineering from the College of Engineering and Technology at the Orissa University of Agriculture and Technology. His technical areas of expertise include COBOL, DB2, CICS, MQ Series, J2EE, .net, and legacy modernization.

Ofer Pharhi is a z/Platform Technical Expert in Israel. He has experience in migration and conversion projects in the z/software field. He holds a B.A. degree from the University of Derby, and a PMP credential from the PMI. His areas of expertise include z/OS system programming, DB2/SQL, COBOL, CICS, Time Sharing Option (TSO)/ISPF, and REXX. He also develops and teaches various z/OS courses.

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author - all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and client satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

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 publications in one of the following ways:

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

  ibm.com/redbooks

- Send your comments in an email 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

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/pages/IBM-Redbooks/178023492563?ref=ts
- Follow us on twitter:

  http://twitter.com/ibmredbooks
- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806
- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm
- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

Part 1

Executive summary: Overview

This section introduces the major theme of this book, which is optimizing your z/OS traditional application development through attention to people, processes, and tools.

Our premise

In the world of software development on System z, we hear a lot about “Enterprise Modernization.” We have heard a lot about this term for over a decade. In fact, we probably think we have modernized, by introducing CASE, SOA, and so on.

And yet, how ironic it is that the tools used by developers to create modernized software—and to enhance and support traditional z/OS applications that were introduced during the 1970s—have changed little since then.

It is even more ironic, given that few technological areas in the world have evolved at a more accelerated pace (since the Gerald Ford administration) than software. Yet, your organization might be in this situation, using these wonderful, but antiquated technologies:

- Time Sharing Option (TSO) and Interactive System Productivity Facility (ISPF) green-screen editors, configuration management, and build management
- Development processes and workflow that are held hostage by 3270-based technologies
- Specific skills required by 3270-based technologies to build business applications

There is nothing inherently wrong with green-screen tools; they still work, adequately, even 40 years later. But a 1970s mainframe tape drive also still works, and we are betting that you do not use tape drives in your data center (or at least... we hope not). Also, on the subject of “adequate”—is it possible that today, development tools are expected to go beyond “adequate”, to the level of “best in class,” meeting requirements standards set by today’s and tomorrow’s workloads—even as they save time and money?

In this book, we hope to show you, with management and practitioner-level technical content, both what the new tools are and how to use these tools successfully to raise your organization’s application delivery standards in the three most common z/OS maintenance usage models.

We want to help your people with better processes and modern tools. We hope you enjoy taking a journey that might save you money, time, and labor, while allowing you to meet the strategic goals that you have set for your company.

1

Executive summary: Value and investment

This executive summary describes the value of z/OS applications and the importance of z/OS application maintenance and support— investing in people, investing in optimizing your development process, the tools, the value of optimizing your maintenance life cycle, and the keys to success.

1.1 The value of z/OS applications and the importance of z/OS application maintenance and support

Well over twenty-five years have passed since companies implemented the first mainframe business applications in the first data centers. Hardware and software, platforms and paradigms, computing languages, methodologies, software journals, tools, technologies, and vendors have come and gone in that time. But, developers have built applications. And, if they built those applications to specifications and if the business processes that these applications support are still viable, the same applications might still run your business.

The applications represent (in accounting terms) “intangible assets” of significant value and intellectual property that automates, extends, and encapsulates other assets of your business. Their book value, if viewed from the standpoint of “income generated over time”, is substantial. Software, like all corporate assets, requires reinvestment to sustain it, for continuous improvement and growth to support your changing business model. This reinvestment is called maintenance.

The following definition is an industry-standard definition of maintenance:

> *“The process of converting, revising or correcting a software application after production-cutover to correct errors, improve runtime efficiency, or adapt to changing business or transactional requirements”*

Maintenance provides the continuity of your business. And just as you take advantage of advances in tools, technologies, and software development processes (or any other aspect of your business), three areas of investment exist that will help you fine-tune the maintenance of the applications that run your business (Figure 1-1):

- Investment in people
- Investment in process
- Investment in tools

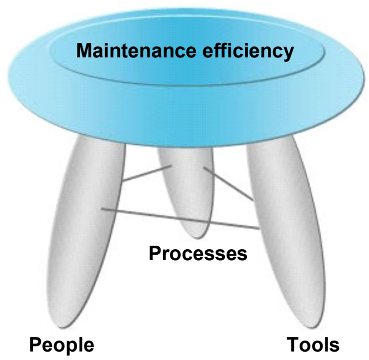


Figure 1-1 Three key organizational areas to renovate

1.2 Investing in your people

Your business either benefits from or is handicapped by the quality of the work produced by your people. And the quality of the software maintenance that is performed on your business applications by your people correlates with the caliber of the three kinds of knowledge used every day by the maintenance staff in your development shop.

1.2.1 Application domain knowledge

Application domain knowledge is the understanding of how *your* software runs *your* business. It is to a certain degree industry-vertical knowledge but to a large degree the unique business rules and processes that differentiate your corporation, setting it apart from competitors in the market.

The trustees of application domain knowledge are the business users. Ensuring that the recurrent analysis needs of the software maintenance staff for authoritative application domain knowledge are satisfied is a fine-tuning key to success.

1.2.2 Technical knowledge

Obviously, your staff must have in-depth mastery of the complete technology environment underlying your application software stack, as well as mastery of the specific maintenance tools and processes that are employed at your development shop. The complexity of production software, especially existing code, is well known. But, historically (and unfortunately), maintenance and support are often seen as less-than-prestigious assignments that are given to those individuals who are not technically up to developing new code.

This approach is a recipe for disaster. Imagine your most novice programmer, on call in the data center when a production run ABENDs and chaos ensues. Providing maintenance teams with premier technical learning opportunities is a fine-tuning key to success.

1.2.3 Application semantics

Application semantics refers to the custom code that realizes the unique business rules and idiosyncratic in-house processes that keep your company running. In most development shops, the original developers of the application have long since moved on or retired. So, learning an application's metamodel and its vocabulary, understanding the coding patterns and application organization, visualizing its data flow and processing paths, being able to predict the state of variable values, and so on become the first and usually the largest challenge your maintenance teams face.

Thus, the value of your software investment as it is maintained over time is determined by the quality and accuracy of the understanding of the application semantics. Providing maintenance teams with static and dynamic maintenance tools that simplify this arcane task is a fine-tuning key to success.

By investing in people, we suggest that you set up a *software maintenance center of excellence* around these three areas of knowledge acquisition, formalizing taxonomies of learning and setting measurable goals, objectives, and so forth. Part of the information is general to all maintenance, such as ongoing education in z/OS skills and infrastructure and any specific education about the tools and processes that you adopt. Other topics are specific to your business areas and the particular code base that a team supports. Specific knowledge transfer includes these tasks:

- Periodic meetings with users and subject matter experts (SMEs) to transfer the domain knowledge of the system
- Time set aside specifically for improving application understanding and for building the kinds of procedural models that individuals need before beginning any code fix or enhancement.

1.2.4 Software maintenance center of excellence

Is setting up a software maintenance center of excellence really necessary?

No, but what if doctors do not bother going to conferences to learn about new medical advances and discoveries. Or, what if auto mechanics are not equipped with (or bother to learn how to use) today's computerized diagnostic tools to tune car engines more quickly and accurately.

It is the collective experience of this book's authors that we have never heard a complaint from a developer about how overtrained or over-prepared the developer was for an assignment or a project. On the contrary, what used to be taken for granted in most companies (technical education that was built into the work year) has predominantly been replaced by Internet-based, on the job, or just-in-time training, which is a euphemism for unstructured or haphazard learning that is informal and almost always ineffective relative to quality, structured education.

There is most definitely a technical and potential loss of business income and opportunity when you use under-trained developers for programming. Consider anyone from a mechanic to a surgeon. How comfortable are you with their work if you know of gaps in their knowledge and capabilities? See Table 1-1.

Table 1-1 Maintenance categories

Category	Immediate work	Scheduled work
Responsive	**Production support** Emergency code remediation due to production outage issues	**Software maintenance** Code remediation due to changes in the business
Proactive		**Code optimization** Efficiencies, standards and documentation

1.3 Investing in optimizing your development process

The need for a methodology or multiple methodologies—one for each of the four distinct categories of maintenance described in Table 1-1 that address how to tackle the diverse and complex issues presented by z/OS software maintenance in a systematic way—seems obvious. And yet, you rarely see a formalized process for how to maintain an application at a certain company. Application maintenance is almost never considered as important as the system development life cycle (SDLC) used to develop new systems.

You might find documentation, such as informal “cookbooks” and other collateral based on lessons learned for performing certain aspects of maintenance work (that is, steps for handling data center ABENDs and on-call situations). And rarely, you might have a true software maintenance methodology for handling adaptive maintenance in your development shop. If so, you are in the minority.

Just like the applications they support, your development shop’s software maintenance methodology, if it exists, falls into the category of intellectual capital. And, the investment in correcting, adapting, and perfecting your maintenance processes, such as revising best practices and including lessons learned in the software maintenance methodology, is a fine-tuning key to success.

An abundance of industry research, white papers, and studies exist about maintenance methodologies that are available from organizations, such as the Institute of Electrical and Electronics Engineers (IEEE), Association for Computing Machinery (ACM), and so forth. If you currently do not have a maintenance methodology, you do not have to start building one from nothing.

It is important to point out that we do not advocate any particular methodology. We do not think that a rigid and complex methodology is the best answer to addressing software maintenance problems. In our experience, if a methodology becomes onerous, or is pushed dogmatically, development teams either find a way out of using it or it becomes another cost issue, not a solution to the problem it was intended to solve.

In this book, we advise that you provide enough systematic structure so that developers do not have to make determinations during projects that end up wasting time or becoming strategic mistakes that cost the organization time and money. You need enough structure to direct developers in emergency maintenance tasks to fix problems that halt data center operations.

One powerful process-specific fine-tuning best practice is to appoint a single person to be the software maintenance architect at your development shop and task that person with the responsibility for establishing organizational controls:

- Developing and refining the maintenance plan
- Publishing a dictionary of your organization’s software maintenance terms and vocabulary
- Creating and managing your software maintenance methodology
- Surveying and analyzing the maintenance problem factors affecting maintenance in your development shop:
  - SME knowledge
  - Maintainability metrics
  - Reconciling problem factors from the user’s versus developer’s perspectives

- Quality of documentation
- Programmer effectiveness, motivation, and turnover
- Age and condition of your business applications
- Integrity of production data
- Machine availability

- Uncovering and putting into practice worthwhile technical insights and empirical data points available from journals, conferences, and so forth

We are aware of one development group, through the research that its Software Maintenance Architect undertook, that found that by “batching” adaptive maintenance requests, the group improved the efficiency and throughput of its projects by almost 20%. Another reason for appointing a single person is that the technical insights and data points are often published in the obscure language of formal research. It will be challenging to find even one person with the right mix of practical production experience and knowledge to fill this role.

1.4 Tools

The software stacks running on and attached to your z/OS have evolved and grown in size and complexity just as budget reductions due to our most recent recession have inversely reduced staff and time. So, from a technology perspective, tools that scale to the second decade of the 2000s are now a necessity.

Using the same green-screen products and associated labor-intensive workflow in the maintenance life cycle that you used twenty years ago only ensures that things are not done efficiently, they are not done to specifications, or they cost more than necessary.

Consider, as an example, how you might debug a situation where incorrect values showed up in a Java™ web page that rendered data from an IBM Information Management System (IMS) database that was accessed through a CICS transaction using a Web Service. How do you track a problem like this one down? Where do you start?

Fortunately, the software industry has moved beyond Time Sharing Option (TSO)-based and manual approaches to maintaining and supporting cross-platform or even complex single-platform applications using the following mature and integrated technologies and products in these areas:

- Application understanding and software maintenance analysis:
  - IBM Rational Asset Analyzer
  - IBM Rational Developer for System z

- Development using a modern integrated development environment (IDE):
  - IBM Rational Developer for System z
  - Plug-ins to Rational Developer for the following tools:
    - IBM Rational Asset Analyzer integration
    - Problem Determination Tool suite
    - IBM CICS Explorer™ and tools for CICS online application work
    - A host of IMS tools for database, such as Data Language/Interface (DL/I), and for IMS online application work
    - DB2 tools
- Team collaboration and source control management:
  - IBM Rational Team Concert for System z

Products that increase the quality and quantity of developer deliverables while simultaneously lowering development CPU consumption been around and are in widespread use. These technologies have proven to save companies time and money and, ultimately, to improve application quality and to lower defect rates. Utilizing tools in these three categories is a fine-tuning key to success.

1.5 The value of optimizing your maintenance life cycle

Why invest in fine-tuning your software maintenance? Research shows that the total investment in maintenance activities is between one and nine *orders of magnitude* more than what is invested in software development and delivery. Even taking the lowest of these estimates, it is clear that incremental improvements in your development shop's maintenance organization, its people, processes, and tools, can easily be the single best financial investment that you make.

Appendix A, “IBM Rational Developer for System z productivity benchmark comparison” on page 587 describes the use of one product, Rational Developer for System z, and its effect on productivity and millions of instructions per second (MIPS). Two recent IBM studies (data points available in this book's appendixes) show that using Rational Developer achieved these benefits:

- Lowered development MIPS consumption by 42%
- Improved developer productivity across a wide spectrum of typical z/OS maintenance tasks by over 30%

In most of the business justification documents that we have seen where companies successfully implemented the products discussed in this book, the return on investment (ROI) break-even occurred within a year, and often within nine months. Of course, financial benefits continue to accrue after the first year, with net savings often measured in millions of dollars over three to five years.

In this book, we offer specific instructions for improving or better utilizing your development knowledge, process, and tools as you continue to manage the complexities of application maintenance.

1.6 Executive summary: Keys to success

Software is the foundation of today's business. While analysis, design, construction, and deployment are all key phases of the development life cycle, software maintenance, which ensues the moment that your application goes into production, and continues until it is sunset, is undoubtedly the most important single stage.

A successful application development shop needs a dynamic maintenance plan, a dedicated maintenance supervisor, and a skilled maintenance team to continually renew and sustain the software, allowing you to exploit its business value and prevent costly data failures, defects, and applications that fail in production:

- Endorse the business value of your production software and the maintenance process of that software as capital assets across your organization.
- Employ a software maintenance methodology that takes into consideration the various categories of software maintenance. This methodology undergoes periodic refinement and renovation, based on industry research and your in-house "lessons learned".
- Establish a position overseeing software maintenance, with responsibility for the establishment of your development shop's maintenance plan, the progress of maintenance practices, and the integration of the maintenance tools and methodologies used, both in-house and outsourced.
- Ensure that the software maintenance staff has easy access to application domain knowledge SMEs.
- Provide software maintenance personnel with access to premier technical education.
- Install key software maintenance tools for application understanding and impact analysis, dynamic code analysis and testing, and team collaboration.

Part 2

The optimized life cycle: Project managers

In Chapter 2, “The optimized life cycle: Project managers and implementation teams” on page 15, we dive more deeply into the topics of people, process, and tools, drilling down another level from the executive summary introduction. We describe topics at the level of the project manager, who is responsible for effectively implementing a better development life cycle and for helping to achieve strategic corporate goals through more efficient tactical application development and delivery.

We also describe in-depth specifically how to manage an implementation for the Rational Developer for System z product:

- Implementation project steps
- Deliverables
- Stakeholders
- What to do
- When to do it
- Who does the work
- What to expect from IBM

We hope that this material, which has been drawn from the authors' direct experience both with the tools and helping clients implement the tools, gives you a clearer understanding of the elements involved in managing both your team using the tools and your team moving toward using the tools.

2

The optimized life cycle: Project managers and implementation teams

The computer industry defines four major categories of maintenance:

- Corrective: Maintenance required due to software bugs and business processing errors introduced in a release
- Adaptive: Maintenance for changed business and transactional requirements
- Perfective: Maintenance to make the code perform better at run time (lowering CPU, I/O, or wall clock time) or to make the code easier to understand and maintain in the future
- Emergency: Solving production ABENDs, which is necessary to keep the systems operational

Other organizations, such as the International Organization for Standardization (ISO), also categorize the types of software maintenance. Overlap and similarity exist among the definitions and terms. We describe all four areas, primarily corrective, adaptive, and emergency maintenance. See the use cases in Part 4, “z/OS application maintenance and support” on page 473. In this chapter, we provide a level of detail about people, processes, and tools that is specific to project managers and implementation teams.

2.1 People

Who are the people that you manage who maintain software? Through our informal research for this book surveying a sampling of 36 z/OS shops from the Global 2000, we identified the following categories and breakdown, as shown in Table 2-1.

Table 2-1 Software maintenance methods

Maintenance team organization	Percentage of shops that use this method
Developers who wrote the code maintain it (primarily)	7
Separate maintenance in-house team	16
Local consultants	8
Offshore or outsourced maintenance	40
Other (various combinations of these methods)	29

When you analyze these numbers, you see that roughly 60 percent of software maintenance is done by external development teams, not by a shop's own technical staff. Fine-tuning software maintenance by individuals who did not write the code before being asked to modify it is a challenge:

- It is a larger imperative to optimize an external development team: people, process, and tools.
- You often can optimize an external development team, because the external team does not have a history with the existing code that might impede technological advances.

2.1.1 Application domain knowledge

Next, we describe the importance of application domain knowledge.

The importance of the business and programmer/analyst

Originally, there were users, system analysts, coders, and problems. The system analysts were overwhelmed, because they were so busy translating business requirements into procedural specifications, flow charts, and Warnier-Orr diagrams. The system analysts needed to understand the business requirements to teach the users. The system analysts and the business process knowledge that they acquired became a development bottleneck.

The industry eventually solved this bottleneck by making the coders that stayed at various development shops or within a domain range of shops, such as insurance in Hartford, CT., into programmer/analysts. The programmer/analysts interacted with users and systems analysts. The bottlenecks eased. And, as the programmer/analysts accumulated business knowledge over the years, their work became even more valuable and higher quality.

In many respects, the current situation is analogous to the late 1960s and early 1970s, with a split between language coders (who used PERFORM) and business/system/programmer/analysts who understand how a particular division of an insurance company adjudicates claims. The key to success for your company is to retain those individuals on staff who have in-depth technical business operational knowledge.

Issues with retaining experienced people

There has been much talk for a long time about how to replace the soon-to-retire aging COBOL population of the world (until this past recession has mitigated this problem to a large extent). But primarily, the issue is not COBOL programming and technical skills. Currently, many issues exist around retaining experienced people:

- Who has the knowledge of the business term (and variable name) coverage-code
- How to successfully implement modifications to a calculation that transacts millions of dollars a day
- How to edit complex business rules
- How to understand which production job can be released in the data center if a job stops and whether there are file update and synchronization dependencies
- Why a particular record value is considered required

This intersection of business/operational and technical/software expertise is priceless.

2.1.2 Technical knowledge

Because so much of the software maintenance is performed externally, you must find ways to certify that the COBOL programmers who make changes to your general ledger programs get more than three weeks of entry-level training. A key to success is for you to assume responsibility for certifying technical credentials, both for your in-house staff but also for any vendor with long-term contracts, especially if you have already reduced in-house technical and programmer/analyst staff.

Many industry-supplied tests and certification exams are available in most of the z/OS disciplines. For example, you can get information about testing at this website:

http://www.iccp.org

Ensure that your own in-house staff has unfettered access to technical learning content and expertise, especially any groups dedicated to emergency maintenance. Understand that we are not proposing a limitless budget of expensive vendor-training classes, but that someone, perhaps the software maintenance architect, is responsible to establish a set of links to the kinds of quality materials available on the Internet. Make access available to IBM developerWorks® (articles, books, and language manuals) and to collections of technical articles:

http://www-949.ibm.com/software/rational/cafe/docs/DOC-3024

Note that an opportunity for saving time and money exists by educating maintenance developers well in the traditional z/OS-specific areas of highest impact:

- Internal data representation problems, which cause a significant percentage of the production ABENDs on z/OS
- Language-dependent issues:
  - Fall-through problems when mixing COBOL sections and paragraphs
  - Flow issues when replacing “Next Sentence” with “Continue”
  - Format conversions, such as date, time, and time stamp math
- All the IBM MVS™ system completion codes and their common causes and solutions:

  http://www-949.ibm.com/software/rational/cafe/thread/2241?tstart=15
- Common business and z/OS application problem areas, such as date arithmetics, Y2K-windowing, static versus dynamic calls, and so forth
- Compiler options and their effects on testing, performance, and debugging:

  http://www-01.ibm.com/support/docview.wss?uid=swg27001475
- Runtime software issues and common complex coding constructs:
  - DB2 SQL problems mixing AND/OR, table joins, and so forth
  - Information Management System (IMS) problems with traversing logical databases and secondary index databases and using multiple program communication blocks (PCBs)
  - Virtual Storage Access Method (VSAM) access method sharing between CICS and batch, file status, and so forth

Note that deep-dive education is required. The existing software *works* and often is an extremely *complex* system that works. Assigning an intern to a production support role for a complex batch system is a serious mistake.

2.1.3 Application semantics

Application semantics is the metadata (that is, information about information), learning the business and technical meaning of text and other data in an application, and learning the connections (see Table 2-1 on page 16):

- For in-house maintenance done by developers who wrote the code, application semantics is not an issue. If it is an issue, you have bigger problems than just software maintenance.
- For separate maintenance teams in your organization and for local consultants, learning application semantics is straightforward. We still advise that you use static and dynamic code analysis tools to simplify and streamline the learning and reduce time and risk. Static and dynamic analysis tools can raise in-depth technical semantics and expose the use of key elements at a much higher level of abstraction than traditional, manual source-based unstructured search activities.
- For offshore and outsourced maintenance, using static and dynamic analysis tools is critical and not only by lead project analysts during the "analysis and scoping" phases of a project. It is critical to have these tools available during technical construction and testing phases of projects.
- Consider the following scenarios:
  - You make changes to a computation. Suddenly, it occurs to you that you are not entirely sure how or from where a variable's value has been derived from the file read to your statement. What do you do at this point? Keep typing? Of course not. You return to the code, program, or application analysis to refine or complete the mental map of the logic that you are revising, before changing another line of source code.
  - Or, you are changing a performed (chained) routine and you need to know that fall-through will not occur or that it will not affect other nested paragraphs or functions.
  - Suppose you are testing a module at the source level and watching as the lines execute in sequence. Suddenly, the FALSE path of an IF statement is taken when you expected the TRUE path to be taken. Do you ignore the problem (your procedural misunderstanding) and keep going? Hopefully, not. You return to structural and program flow analysis.

Every developer who has ever modified code has experienced these scenarios. Without static and dynamic analysis tools, your only choice is to use manual Interactive System Productivity Facility (ISPF) FIND commands and to read through program listings. Yes, it can be done, but it is much slower and much more expensive. And, it can be done only by substantially more expensive experienced developers. Use the static and dynamic analysis tools to regain control over the key area of application semantics software maintenance.

2.2 Investing in the process

Figure 2-1 on page 22 shows a typical process flow. It organizes the software maintenance process or life cycle in seven steps (following a maintenance or support request):

- Problem taxonomy: Classifying or categorizing the work to be done at a high level
- Project scoping: Determining personnel requirements, the level of complexity, critical path, project milestones, and so forth
- Analyzing/designing: Understanding the issue at a production-quality level of veracity and software engineering precision
- Developing/building: Technically designing and constructing the solution
- Regression testing: Determining that the new functionality works in the new release and that nothing else in the application has been broken in the new release
- Quality assurance and acceptance testing: Sometimes includes performance testing as well, final user testing before the production cutover and establishment of the next version's regression testing baseline
- Production cutover: Moving all the modified software elements into production

You might recognize that several or all of these steps occur in your shop's approach to software maintenance. Or, you might have similar, fewer, or more steps, depending on the formality of your methodology and how it is applied by various management teams. Your process model might contain much more detail than Figure 2-1 on page 22.

For example, your process model might contain these elements:

- Input and output deliverables
- The activities grouped
- Related and supporting subprocesses
- The control
- A set of metrics

If you have not built or acquired your own formal maintenance process, use our steps as a starting point.

Here we offer three points on the maintenance process:

- It is the experience of most of the shops that we contacted for this research that the needs of each category of maintenance (emergency, corrective, adaptive, and perfective) expand out to additional subprocesses for most all of the steps in the process (Figure 2-2 on page 23).
- Your shop's own experience, priorities, and prerogatives, and even your approach to implementing methodologies always supersede a generic, one-size-fits-all approach.
- Working toward a rigid, structured, and complex procedural methodology might be an indulgence that will not provide sufficient returns on your investment as anticipated and must be considered in the context of your shop's inclination toward results versus process-driven activities and project management.

Also, notice in Figure 2-1 that we have positioned the IBM Rational products that support the life cycle next to the stages in which they are used. Note that we use IBM Rational Team Concert for System z (RTCz), which is a team collaboration, task management, build, and configuration management product, and IBM Rational Developer for System z (RDz) throughout the life cycle. These products represent the foundational, Eclipse-based integrated development environment (IDE) that is used in all daily work by all practitioners for all steps in the life cycle.

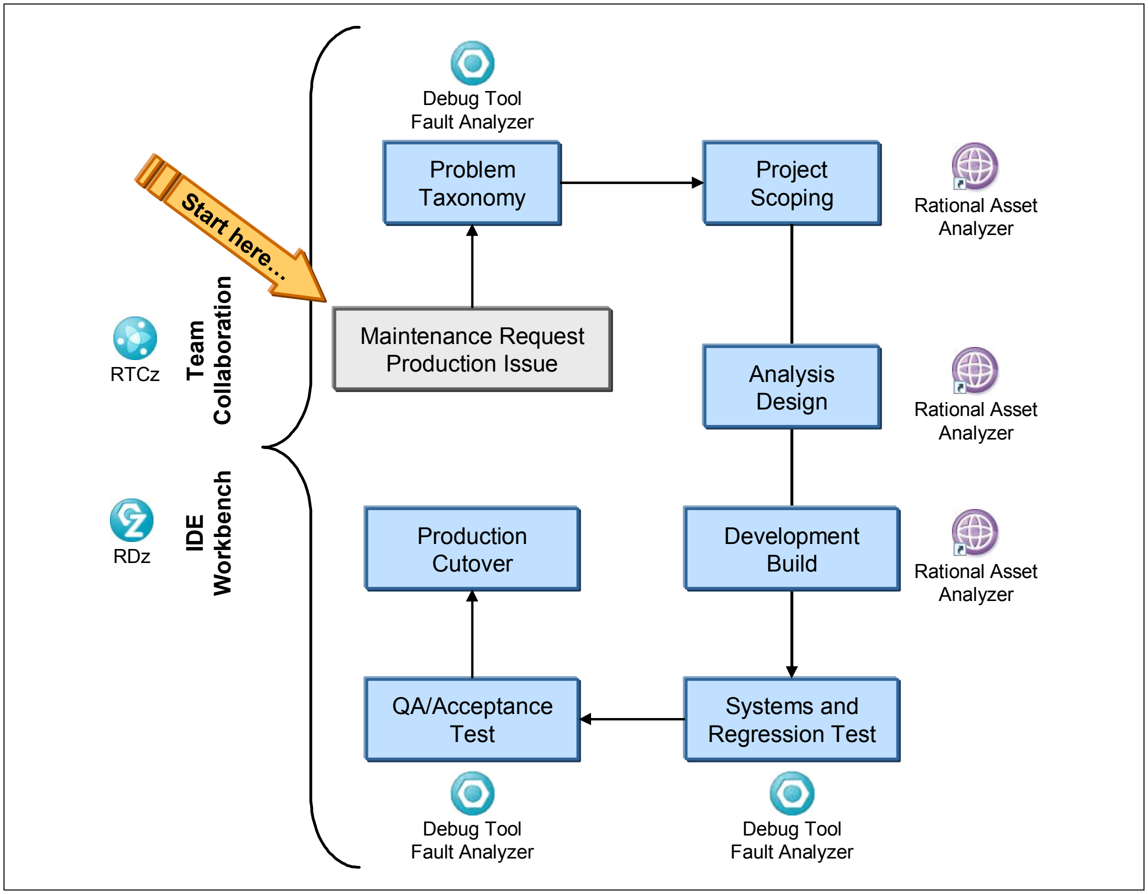


Figure 2-1 A standard software maintenance life cycle

2.3 Tools and the process steps

When thinking about the required, optional, and effective use of tools throughout the software maintenance life cycle, consider that there are two categories:

- Tools that bring value to each step, which include cross-platform collaboration products and source control management tools
- Tools that have step-specific advantages, such as the static and dynamic analysis tools

Your shop might create its own maintenance-tools-process-driven matrix, which will change over time, as new technologies and new problem domains appear, but Table 2-2 shows a typical correlation of software maintenance life cycle phases and tools that can help fine-tune your efforts to save time and money. In the next section, we describe the tools, functionality, and features applied to software maintenance.

Table 2-2 Tools used at specific life cycle stages

Tools Used at Specific Lifecycle Steps		Tools Used Throughout the Lifecycle
Problem/modification identification, classification, and prioritization	**Dynamic Testing tools** – for initial breakdown of emergency and corrective maintenance problem definition.	**Team collaboration** **Source Code Management**
Project scoping analysis	**Dynamic Testing and Static source code analysis tools** – for trustworthy/accurate results and efficient/cost-effective analysis activities.	
Design	n/a	
Implementation construction	**Static source code analysis tools** – for solving inconsistencies in developer understanding.	
Regression testing System testing	**1. Dynamic Testing tools** for revealing problems during test & debugging. **2. Static source code analysis tools** for solving problems, stemming from an inconsistent developer understanding (which – if you think about it, is the source of the preponderance of software problems).	
Acceptance testing	**Dynamic Testing tools**	
Delivery	n/a	

First we discuss why tools are necessary, and then explain which tools to use.

For decades, software developers looked through listings using paper clips and sticky pads when they needed to bookmark variable references, or on z/OS, they used ISPF search and labeling operations, and eventually code went into production and worked. Next, we describe the value proposition for using the four categories of tools that are shown in Table 2-2 and discuss the must-have features that offer enough value for you to invest in them and use them.

2.3.1 Source code management

Source code management is the easy sub-category. Considering the size, complexity, and dynamic nature of modern z/OS development scenarios, especially with the introduction of offshore development, it is taken for granted that you must have a robust source code management (SCM) product for your development and maintenance activities. This SCM product automates the version control process and offers revision management, file locking, the ability to return to previous versions (by stripping out deltas), and version merging.

It is common for multiple versions of the same software to be deployed at separate sites, but for fixing bugs and for adaptive maintenance, automating the control of access to source versions is a long-established best practice. You likely already have SCM up and running at your shop, and you probably have customized it extensively, as well.

2.3.2 Team collaboration software

Even if you are well versed in SCM, you might not be familiar with team collaboration software, which is a software dashboard with these functions:

- Allows project leaders to define, organize, and track your software maintenance, support, and development teams and their projects (not program source, people, and processes)
- Provides simplified interaction and communications between subject matter experts (SMEs) and the technical programmer/analysts and architects who interact with them. This interaction can be captured electronically for reuse, refinement, and documentation.
- Accommodates management reporting and governance of project milestones and deliverables
- Enables project technical developers to share source and model-based development artifacts in the context of performing project work

By raising the team concept to the level of an automated entity and providing integrated workflow among all stakeholders, software maintenance projects (across all four categories) are freed (to degrees) from repetitive, manual tasks, and allowed to focus on the business and technical problems. Think of team collaboration software as reducing the costs of administration, which can be a significant cost and barrier to productivity enhancements in an organization.

Regarding the challenges introduced by offshore maintenance projects, team collaboration is well positioned to solve the latency, communications, and logistics issues that are part of worldwide (or even across town) shared project efforts. A distributed development environment (versus a central development environment) creates organizational challenges that must be addressed to gain the understood benefits.

2.3.3 Static and dynamic code analysis tools

Both static and dynamic analysis tools assist developers in "gaining intellectual control over an application" per Larry England, IBM, Santa Teresa Labs.

Research has shown that there is an increase in cost in productivity when maintaining large systems. This lack of economy is due to the large number of variables and functions of interest (often in the hundreds to thousands) scattered throughout large amounts of software (often in the hundreds of thousands to millions of lines).

Historically, only experienced z/OS developers with over five years of production experience are capable of building precise, clean, and faultless internal/mental maps, thereby gaining "intellectual control" of the code execution paths, key variables, their state transitions, and so forth.

Over the past two decades, many of these individuals have retired or moved on to management positions. Now, many maintenance teams are in the difficult position of using relatively inexperienced developers to make technical judgment calls about changes to core systems, which have gotten older, more complex, more brittle, and harder to understand. This situation represents a remarkable level of risk. All the keys to success are cumulatively helpful, but electronic automated analysis software consolidates and amplifies your savings of time and money significantly.

Static code analysis

One of the unintended consequences of Y2K was the birth of static code analysis tools, which were successfully used in Y2K adaptive maintenance, because January 1, 2000 occurred with virtually no major z/OS meltdowns. Every category of software maintenance relates to an aspect of the Y2K problem.

Therefore, if the use of static analysis tools was the central key to success for finding and fixing century date routines pervasive in z/OS applications, applying static analysis tools to corrective, adaptive, preventive, and perfective maintenance makes business sense. And, passing up the opportunity to employ them means making up for their automated and comprehensive features with additional time (a not insignificant percentage increase), risk (through an incomplete or faulty analysis), and the money associated with additional time and risk.

Features

The information needs of maintenance and support staff vary by the type of application learning and comprehension model employed. It is commonly understood that you "learn" a program either top-down (by studying documentation, talking with SMEs, and reading the functions and procedures starting from the entry point, one logical branch at time) or bottom-up (starting with a specific function, record, or variable, often working from program listing cross-references and tracing backward and upward). This approach is meticulous, difficult, and error prone, and it is also the nucleus of a software maintenance project. In reality, for projects or systems of significance, programmers switch between the two approaches, integrating information into a final mental model of the application's execution and how it relates to the current business or situation.

Tools, then, that comprise features automating or simplifying top-down and bottom-up analysis fast-track your teams to improved productivity and lowered risk. At a minimum, these tools need to contain the functionality that is described in Table 2-3 and come with a graphical, mouse-based IDE, largely because the task of understanding code is keyboard-less. Navigating through diagrams and hyperlinks with a mouse pointer instead of typing FIND commands into green-screen panels is actually a critical element of the usage model.

The application semantics or metadata (that is, information about information) surfaced by static analysis must be contained in an easily searchable dictionary or database. The tool's metadata, extracted and exposed as structured top-down diagrams and bottom-up element-level reports, which are cross-referenced and hyperlinked, must be built from the source code to provide the maximum semantic integrity.

The list of requirements in Table 2-3 is not comprehensive. A good static analysis tool can assist in answering the majority of the questions in software maintenance projects.

Table 2-3 Use cases of optimized life cycle tools for static code analysis

Development Use Case	Description	IBM/Rational Solution
Top-down systems views	Iconic/Graphical images – and detailed reports, that correlate to system flow charts.	Rational Asset Analyzer
Hyper-linked transaction and call/calling program views	Execution chains abstracted up to a graphical level with accompanying detailed report analysis. Hyper-linked for top-down study.	Rational Asset Analyzer
Hyper-linked batch job views	Individual steps in a job (“run-unit”), graphically exposed and hyper-linked, with sub-elements that show (at a minimum) programs, utilities, external files, etc. used by each job step. Accompanying detailed report analysis. All hyper-linked for top-down study.	Rational Asset Analyzer
Hyper-linked Data Flow Diagrams	Hyper-linked graphical depiction of file usage throughout a batch job or transaction.	Rational Asset Analyzer
Program or Module “Functional Decomposition”	Different views or slices into programs breaking the control flow logic up into more granular.	Rational Asset Analyzer Rational Developer for System z
Impact Analysis	“What If” analysis, before source modifications are made – at all levels of scope: Program, Run-Unit, Transaction, System-wide.	Rational Asset Analyzer
“Where Used” and “How Used” Analysis	Ability to isolate variables and reveal only: statements that change the variable value, statements that test or reference the variable value, declarations, etc.	Rational Asset Analyzer Rational Developer for System z
N-way search capabilities	Search in meta-model for elements. Search in source for elements and element subset.	Rational Asset Analyzer Rational Developer for System z
Customized code windows	The ability to create on-the-fly combinations of editable windows in order to provide the best source-level perspective an area of code.	Rational Developer for System z
Subset capabilities	Ability to group (subset).	Rational Asset Analyzer Rational Developer for System z
One-click Access declaration of variable	Ability for the tools to “keep up with the developers exploratory process”.	Rational Developer for System z
Expand/Collapse code sections	Isolate to manageable code blocks.	Rational Developer for System z
Extensible Query Language	The ability to compose completely-custom queries against the meta data in the tool’s dictionary or database.	Rational Asset Analyzer
Integration with the “Development Workbench”	The ability to use the results of all of the above analysis reporting directly inside the edit/compile/debug tools.	Rational Asset Analyzer Rational Developer for System z
Code Quality Metrics or development “Best Practices”	Show range violations (bad GO TO structures), recursion detection. List inefficient variable declarations, etc.	Rational Asset Analyzer Application Performance Analyzer
Perfective Maintenance	Finding dead code, unused copybooks, etc.	Rational Asset Analyzer
Support Emergency Corrective Maintenance analysis	Show “where modified” code to trace down root-causes of S0C7 and S0C4, etc.	Rational Asset Analyzer
Code slicing and Business Rules Mining	Separating business logic from persistence, data access and presentation logic.	iLog/JRules Rational Asset Analyzer

Dynamic code analysis

Even though static code analysis is an indispensable aid in helping to solve certain problems of software maintenance, especially in the emergency corrective maintenance category, you need another view of the production application. *Dynamic code analysis* is the view that is provided by actually running the live code at the source level and following the execution sequence.

Dynamic code analysis tools allow you to inspect the following components at the detail level:

- The instruction sequence: Every statement as it is "live" is stepped through and animated, running against the software stack that is the deployment platform.
- Variable values: Any values in program storage, file buffers, database records, and temporary storage, to which the code has addressability, are examined.

This type of real-life monitoring at the source code level is unrivaled for solving many of the most obstinate and bewildering software analysis problems and is invaluable for corrective maintenance in every form.

Table 2-4 lists many of the more prominent and useful dynamic code analysis features. Like source code management, it is likely that you already own a product that can be used for dynamic code analysis. But, are you taking advantage of these tools for dynamic code analysis and not only for debugging? Table 2-4 lists many of the ways that dynamic code analysis can provide teams with automated assistance in areas that are traditionally labor intensive and error prone due to the size and complexity of z/OS systems.

Table 2-4 Use cases of optimized life cycle tools for dynamic application analysis

Development Use Case	Description	IBM/Rational Solution
Dynamic, source-level step-by-step animation	Iconic/Graphical images – and detailed reports, that correlate to system flow charts.	Debug Tool
Present variable values: **- Characters** **- Hexadecimal**	Execution chains abstracted up to a graphical level with accompanying detailed report analysis. Hyper-linked for top-down study.	Debug Tool
Interrupt execution based on convenient set of options:	z/OS applications are almost always too large to step through from their beginning. Various break points must be set in order to make the tool usage practical: Entry to module, value change, conditional value, etc.	Debug Tool
Selective subset of Variables to "monitor"	z/OS applications have far too many variables to monitor all at once, and viewing just the variables on the current line is too restrictive. You must be able to make persistent lists of variables for monitoring.	Debug Tool
"What if" Analysis	Besides passively viewing the code execution flow and variable values at run-time, developers often need to modify storage values, alter the instruction sequence, iterate over complex logic, in order to really understand the algorithm or code pattern.	Debug Tool
Access to z/OS Storage and Registers	For debugging assembler programs, or for analyzing a failed instruction at the machine code level.	Debug Tool
GUI Integration with the development Workbench	Far from cosmetic, graphical, integrated Workbenches that include Dynamic and Static Analysis tools are simply expected. To understand this, consider how often you open DOS prompts and type: **copy c:\directory1\subdirectory1\myfile.dat to d:\directory1\subdirectory1\myfile.dat**	Debug Tool and Rational Developer for System z
List programs called at run-time, during a specific transaction	When learning an application (adaptive maintenance) or studying the root cause of an ABEND situation (corrective maintenance) it is acutely important to understand the precise execution route.	Debug Tool CICS Interdependency Analyzer
The sequence of transactions within an application	When learning an application or studying the root cause of an ABEND situation it is acutely important to understand the precise execution route.	Debug Tool CICS Interdependency Analyzer
List files and databases (Read or Updated) used at run-time during a transaction	Following the chain of input/edited/output files as they are read from/written to throughout a transaction or batch application is one of the most common maintenance strategies. Simplifying/Automating this process alone can be worth the price of any product.	Debug Tool CICS Interdependency Analyzer
For Emergency (ABEND) Situations, provide initial debugging data points	Integrated "fault analysis" where valuable research information, such the exact statement in a program that caused an ABEND (and surrounding code) plus the values in the variables under consideration can be critical in timely diagnosis and correction of the problem.	Debug Tool + Fault Analyzer
List program lines not tested – as a result of Regression Data	A "test coverage" utility, that shows marked source of lines tested – and lines untested.	Debug Tool
Integration with static code analysis	The ability to research problems alongside their discovery promotes productivity, cuts down on errors, etc.	Rational Asset Analyzer

2.4 Implementing the tools in the optimized life cycle

As a project manager, you might be heavily involved in implementing one or more of the tools in the optimized life cycle. To assist you, we include a standard or typical high-level project plan that is documented in the remaining sections of this chapter.

Note that this project plan is a generic model, which is based on no single company's input or approach. There are numerous steps in the process that must be customized according to your specific shop and its people, processes, and tools.

2.4.1 The process

As seen in Figure 2-2, organizations typically follow a multi-step, hierarchical process when introducing a technology, such as Rational Developer, into a Time Sharing Option (TSO)/ISPF shop. You must manage this implementation project in a structured fashion, just as you do for any mainframe integration project, with milestones, operational analysis, management reviews, knowledge transfer, and so forth.

The tools of the optimized life cycle are not necessarily a fit for every shop, at this moment in time. But, they offer so many benefits to companies performing both traditional z/OS maintenance and enterprise modernization that deciding when and how to implement the tools needs to be based on data driven decisions not subjectivity and personal bias.

The next few sections of this chapter describe how to approach implementing the tool suite of the optimized life cycle. We also describe how to arrive at an objective and empirical rollout plan that is driven from where your shop currently stands with its application development architecture to where it wants to go to produce business application functionality better, less expensive, and faster.

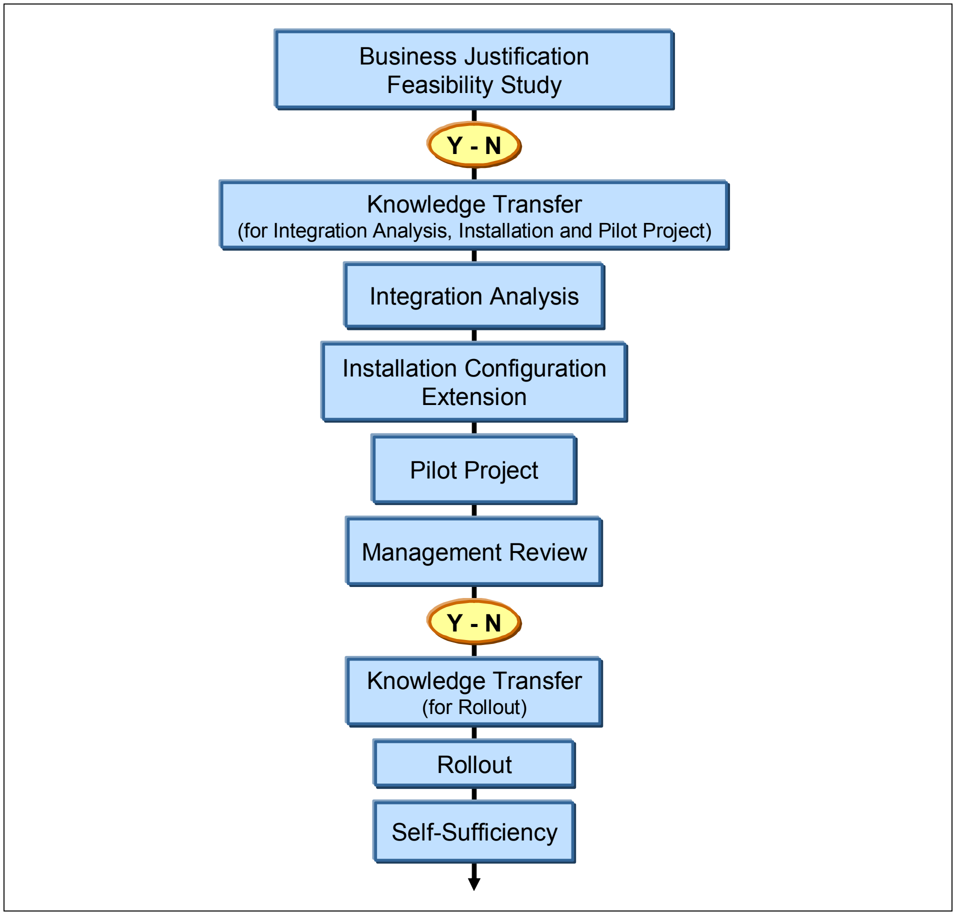


Figure 2-2 Process for implementing Rational Developer tool in the optimized life cycle

2.4.2 Business justification and feasibility study

In creating any adoption strategy for new technology, you must complete the following tasks:

- You must be certain that the optimized tool suite can work in your development environment, with your developers, and at this time. These considerations are the technical factors, and they become part of a feasibility study or a high-level technical evaluation.
- You must build an initial business case, including the following tasks, to provide business and financial justification for moving forward:
  - Relating the tools purchase to business objectives and financial goals
  - Setting up the measurement criteria that is used to determine that the tools can meet your business and financial objectives

Business objectives

Many reasons exist to move to the tools of the optimized life cycle. Certain reasons are purely financial, such as lowering the CPU consumption on the development logical partition (LPAR), and other reasons are business strategic, such as attracting and keeping new employees in the organization (who will resist working with 40 year-old development tools).

But irrespective of your aspirations for the new tools, it is incumbent on your organization to develop and refine the business objectives before proceeding with an implementation plan, if for no other reason than to communicate your investment justification with upper management and chief information officer positions.

And, because you cannot manage what you cannot measure, begin with the discussion about evaluation criteria, key performance indicators, and measurement parameters.

Where to start

Typically, you begin with a product briefing from your IBM Rational account team. This briefing consists of a discussion of your current application development architecture, short-term, medium-term, and long-term plans for your business technology, and problem domain. You discuss whether the tools of the optimized life cycle can be of significant enough value to warrant further discussion, analysis, and evaluation.

During these briefings, you focus on four goals:

- Determining if the optimized life cycle tool suite is technically feasible. You base this decision on your current z/OS (systems) software install stack.
- Evaluating to what degree your development staff will embrace the tools and use them effectively. You can determine (at this initial level) whether your team will accept the tools by seeing one or more deep-dive demonstrations by IBM.
- Preparing success criteria for a pilot project. You need to formalize, externalize, and communicate your success criteria to all stakeholders, including IBM, before the pilot project. This success criteria is the agreement or document of understanding for the pilot.
- Formulating financial return-on-investment (ROI) factors

Financial goals

You can formulate ROI in several ways. You typically measure the following items in a spreadsheet:

- Product costs:
  - Software: Initial and annual support fees
  - Any workstation hardware upgrades: Necessary to run the tools on your developers' personal computers, for example, additional workstation random access memory (RAM)
- Services costs
- Current operational and software development costs:
  - Developer salaries, including benefits
  - Operational development costs (TSO connect time charge-back and so forth)
- Amount of time spent developing code, broken out into these categories:
  - Analysis
  - Edit
  - Syntax check, compilation, or both
  - Test
  - Other
- Expected reduction in operational development costs: Factored into the amount of time that is spent in each development phase
- Expected gains in productivity: Across the development phases
- Productivity increases: Obtained through many factors

Your application development staff must gather most of the spreadsheet information through questionnaires. It might seem like a lot of effort, but without accurate research, it is impossible to accurately determine ROI, because the savings will be expressed as a relative reduction in development costs and a relative increase in productivity. IBM has created a spreadsheet that can form the foundation of your work in this area. Contact your local account team for a copy if you want to use it.

You must also consider addressing the following issues to help meet your strategic goals:

- Attracting and retaining top-level newer employees by providing them with a state-of the-art, graphical user interface-based development environment.
- Modernizing your applications: Providing development staff with tools that allow them to perform next generation technical development work, such as service-oriented architecture (SOA), CICS Service Flows, IMS SoapGateway, CICS Web Services, and so forth. Rational Developer provides wizard-driven development tools for performing this kind of development.
- Standardizing all development tools on an integrated Eclipse platform: Often, it makes sense to provide one toolset that has quality cross-product and cross-platform integration to all traditional z/OS and distributed development staff. This standardization fosters a sense of community and provides benefits in areas, such as collaborative testing (for example, where a Java 2 Platform, Enterprise Edition (J2EE) or Web 2.0 front end calls an IMS or CICS online back-end application).

When IBM meets with your team, your systems programming and database administrative staff will discuss your current z/OS software stack and your practices and schedules for updating your z/OS software stack.

You need to look carefully at the following items:

- z/OS system software release levels: Rational Developer requires a minimum level of z/OS, REXX, ISPF, IBM Software Development Kit for Java, and installation
- Supported and unsupported technologies for which you want to use Rational Developer, in two levels; full tooling support and ISPF-quality support
- Workstation hardware and software minimums
- z/OS compiler (language) and run time (IMS, DB2, and CICS) release levels
- Security package (Resource Access Control Facility (RACF)/Computer Associates Access Control Facility 2 (CA ACF2) or Top Secret) and release levels
- TCP/IP and UNIX® system services release level

From this information, you can quickly ascertain whether the tool suite will install successfully. You will also want IBM specialists to spend at least 1/2 a day demonstrating the tools using various use cases that are relevant to the projects, tasks, and work done by your z/OS traditional maintenance staff. It is extremely important to see a “live” product demonstration, which provides a better feel for the following capabilities:

- Ease of use: Remember that time is money.
- z/OS affinity: Determine how easy it will be for your ISPF development staff to pick the tools up and run with them.
- Flexibility and limits: Using the tools, as they are shipped, see if you can do everything you need to do or are expecting from the products.
- Extensibility: Consider if you can customize aspects of the tool, for example, the manner in which Rational Developer interfaces with your SCM system.

Proof of Technology: An initial human factors evaluation

It is important during this initial feasibility study phase to make a high-level assessment about your developer base and to determine if your employees can transition to an Eclipse-based development tool.

IBM and its certified IBM Business Partners periodically, and also on demand, offer a no-charge Proof of Technology where your programmers work hands-on with Rational Developer in VMware-imaged systems (where all the product installation and integration work is done).

By following a detailed script, during a Proof of Technology, developers can test and evaluate these functions:

- Experience Eclipse-based development
- Analyze, modify, and syntax-check z/OS traditional programs in COBOL or PL/I
- Create Web Services and DB2 stored procedures

Developers can get a sense of the advantages and disadvantages of performing their work with the new tools. You must ensure that your pilot team gets to actually work hands-on with the tools of the optimized life cycle at this point. From the pilot team’s feedback, you can gauge the prospects for adoption and successful use at your company. In addition, developers might help you discover areas of the tools to exercise during the pilot project.

High-level technical evaluation

A technical feasibility plan is the same for the tools of the optimized life cycle as it is for introducing any other z/OS software into your shop. You need to organize an internal evaluation team that is staffed by various technical roles from the following organizational competencies:

- System programmer
- UNIX Systems Service programmer
- Security administrator
- TCP/IP systems staff
- Application development manager or designee
- Database administrator
- Project manager and select members of your technical programming staff: Ideally, a cross-section of veteran lead technical developers and entry-level programmers (two distinct programming demographics)
- Technical analyst, application architect, or “tools implementation” team for these groups:
  - z/OS applications: For evaluating and supporting the use of the tools for traditional maintenance and support
  - Distributed applications (J2EE, Web 2.0, and so forth): For evaluating and supporting the use of the tools for SOA and enterprise modernization
- The pilot team members: Ideally, a cross-section of developers that is representative of your teams (now and future):
  - TSO veteran developers
  - Entry-level programmers
  - A project manager or project lead for the pilot team

Stakeholders

The stakeholders in this process include your technical team and the IBM account team (possibly with an IBM Business Partner).

Deliverables

The deliverable from this process is a high-level feasibility study, based on Rational Developer’s support for your software stack. In addition, your IBM team or IBM Business Partner team learned more about your people and their skills and readiness and your applications, tools, and processes through one-on-one interaction between the instructor and the class (even if done remotely).

2.4.3 Pilot project installation and configuration knowledge transfer

Knowledge transfer plays a large part in the overall key to success equation in product evaluations and yet is often relegated outside of the workflow for the evaluation team and treated as an *afterthought*.

It is our experience that, in actuality, knowledge transfer occurs continuously as part of this implementation process and throughout this implementation process:

- Your developers and systems teams learn about the tools
- The IBM team and IBM Business Partners learn about your shop's unique approach to software maintenance and about your software stack
- Formal and specific information that must be learned prior to each step in the process
- General day-to-day two-way learning that goes on
- A formal platform-transition knowledge transfer process that we describe in 2.4.8, "Knowledge transfer (for the product rollout)" on page 53

Considering that potentially millions of dollars in saved expense and additional business benefits associated with moving your technical staff to a modern software development paradigm are at stake, it makes sense to ensure that the evaluation team and the extended team (the pilot project developers) get fully informed about the product, its strengths, weaknesses, limits, and advantages over ISPF.

Without in-depth knowledge transfer, one of two negative outcomes can occur:

- Developers accept the product based on a narrow or incomplete understanding of the product composition and limitations.
- Developers reject the product based on a narrow or incomplete understanding of product capabilities and the mind set that "What the user does not know how to do becomes what the product cannot do".

The antidote for both of these problems is both proven and simple: in-depth technical training, delivered ideally on-site, by qualified technical educators. Or, if logistics and budget have not yet been allocated for products that are in the evaluation stage, at a minimum, you must obtain deep-dive education for the "tools implementation" and pilot project team remotely through a form of distance education or through self-paced training.

On-site education is always the best choice for this important phase. Having an instructor available in real time to explain specifics can make the difference in acceptance or rejection by someone new to modern IDEs.

You might also want on-site education for these other reasons:

- Customized education that is tailored to your shop's software stack specifics or to the pilot project's application requirements
- The ability to extend a discussion into any area of interest, and there are always areas of specialized interest beyond a standardized curriculum to which an on-site instructor can respond better than any other education offering

Performing an evaluation of the optimized tool suite with unprepared individuals yields a predictable and unreasonable result that is unprofitable in the future for your company. Make sure that you get your people trained before they start using or implementing the product.

Training scope

Certain shops like to train their developers on the entire product to give them a sense of the breadth of the capabilities. Other shops prefer to narrow the focus of the training to what is specifically going to be accomplished in the Proof of Concept (PoC). We think the latter approach is best and makes the best use of the pilot team's time.

Start with a deep-dive session on z/OS application development basics. Be sure to include any training that is needed for the functionality for the pilot project use cases. By deep-dive training, we mean ensuring that individuals understand how to do everything they currently are able to do with ISPF as an IDE. Also, individuals need a solid foundation in the functions that Rational Developer offers that ISPF cannot do, for example, using Eclipse windowing technology for optimized split-screen work.

Just-in-time training is extraordinarily important, because a development IDE, such as Rational Developer, is only locked down and assimilated by new users if they proceed directly from training to practical application and real-world use. If weeks go by between the end of your training session and the beginning of your project training, participants can lose up to 75 percent of their knowledge transfer.

Internal technical and business advocates

It is extremely hard to achieve goals of any substance without leadership. As the project manager, it is your task to lead, orchestrate the implementation process, prioritize requirements, assemble the key players (internally and from IBM), call meetings, document results, understand and communicate issues, and negotiate (internally, as well as with IBM).

Additional leadership exists in the form of technical and business sponsors or individuals within your company that agree with implementing the optimized tool suite and encourage and influence other people toward their vision of ROI or technical benefits. Finding and making these internal influencers part of your team can be a key part of succeeding when the inevitable obstacles occur, for example, when the entire pilot team gets reassigned due to changes in business strategy, or your data center holds a moratorium on new software installations due to staff reductions.

The *business advocate* is typically an officer-level individual that sees the benefits of progressing out of outdated technology that restrict the introduction and development of new business and technological models.

The *technical advocate* is a senior staff member that is an authoritative and respected individual who is sought out for answers to questions and solutions to complicated issues and problems. Ensuring that this type of person is on your implementation team can be critical to winning over established Time Sharing Option (TSO)/ISPF developers that are resistant to moving to a modern IDE.

This process creates the pilot project team and a feasibility study that takes into account and analyzes, assuming a technical fit, the following information:

- A description of the operational goals of the implementation
- A plan for integration analysis and the remaining phases of the implementation strategy
- The selection of the pilot project application
- The selection of the scope of the pilot project
- The formal success criteria for the pilot project
- An ROI document

Training

You must plan for the training process:

- Your tools and implementation technical team and the pilot team will receive training
- IBM or a certified Rational Developer IBM Business Partner will provide the training

Results

The process results in the following teams:

- A project team that is trained just-in-time to begin the human factors analysis segment of this project

- An IBM team or IBM Business Partner team that has learned more about your people, your applications, tools, and processes

2.4.4 Integration analysis

In parallel with or possibly immediately after completing knowledge transfer, you need to bring in one or two IBM technical application specialists to perform an assessment of how to integrate the optimized tool suite into your current environment:

- Development process
- Foundation development software configuration

The concept is simple. The tool suite of the optimized life cycle must integrate into your existing application development process and its best practices and your application development software stack. The tool suite must complement your tools and bring value over what currently works well for your teams.

For example, Rational Team Concert has several large-grain product features, including collaboration, requirements governance, and source control management. Perhaps your shop can benefit from the integrated collaborative features and the requirements governance but not the SCM features, from all three (especially if implementing a new SCM can save you annual license fees), or from just one. You must decide and you can only make this decision by understanding which of the features that Rational Team Concert for System z offers is right for you.

For Rational Developer, the importance of integration is amplified, because based on our experience, every shop has a large number of custom ISPF-based tools and runs its own custom build process. Therefore, understanding the Rational Developer Remote Syntax Check, Compile, Link, Bind, and so forth is critical and most likely dictates a level of customization work even before running a pilot project.

Certain shops have created their own custom (in-house) pre-processor that is invoked prior to compiling on the host. It is usually not difficult to integrate these software components, but obviously, this integration is on the critical path.

There are other necessary integration points, both small and large. As an example of a large integration point, you must connect Rational Developer with your host SCM if you choose not to go with Rational Team Concert for System z. You can expedite this process by using several of the Rational Developer built-in features, such as Developer for System z Common Access Repository Manager (CARMA), which is described in 4.5, “Rational Developer and source control management integration through CARMA” on page 148.

It is important to understand the distinction between integration necessary to prove the concept of the optimized tools during the pilot project versus integration necessary prior to moving ahead with full-fledged production rollout. Full-fledged production rollout integration must be a separate formal project by itself.

An example of a full-fledged production rollout integration project might be to fully merge your mainframe build processes, including your customization to third-party tools, such as Computer Associates CA Endevor, into the Rational Developer workbench as graphical build facilities.

The following points are examples of small integration points:

- Deciding on the role Rational Developer templates and snippets play in the pilot project
- Considering which workspace editor and editing “look and feel” to use
- Determining which Eclipse plug-ins and add-ons will be part of the pilot project

You must perform a software integration analysis prior to the pilot project. The analysis is driven by the goals of the pilot project, not the goals of the production rollout.

IBM Rational Asset Analyzer differs in this respect, because as a static analysis tool, there are only a few integration decisions to make, such as how to transfer the application source code to the repository, how to synchronize updates, and so forth.

Stakeholders

The following individuals and teams are the stakeholders in the integration analysis:

- Your tools and implementation technical team, who lead meetings, describing the composition of your software development architecture
- Your application development analysts or lead technicians, who document your software development process in detail
- IBM application analysts or a certified Rational Developer IBM Business Partner, who take the input that is provided by your team and provide consultative guidelines about what aspects of the tools fit within your best practices

Workspace backup and recovery

As part of the integration analysis, you need to work through the details of backing up and restoring each developer’s workspace. You will learn more about workspaces in upcoming sections, but every Eclipse (and therefore) Rational

Developer project is developed under the umbrella of what is called a *workspace*, which resides on the individual's desktop and consists of this information:

- Preferences
- Local source code, files, data, and so forth
- Connection details and other development artifacts

You most likely want to take periodic backups of every developer's workspace. Certain shops take backups nightly, other shops take backups weekly, but to plan for workstation disaster recovery, make sure that you have built into your plans the workspace backup and then prototype the process through to recovery.

Packaging the installation and automating fix packs

Another aspect of integration analysis is deciding how you package the initial installation of the client workstation software, and perhaps more importantly, how you automate the application of fix packs (software updates from IBM) over time.

IBM has its own method of installing Rational tools and applying fix packs, which is called *Installation Manager*. However, many shops prefer to use a *silent install*, which is an automated process that pushes software out to user desktops in a controlled way.

One additional consideration for this analysis is the element of plug-ins. A *plug-in* is a custom feature (either from IBM or from a third-party Eclipse development shop) that plugs into the Eclipse framework outside of the default product installation process.

Plug-ins are often appealing and seemingly a benign and easy way of adding value to this open architecture. However, when it comes to rolling out a consistent image and especially when it comes to updating and upgrading individual client desktops, plug-ins carry the following additional requirements:

- Test that the plug-ins work with the subsequent release
- Test that plug-ins do not harm other functionality with subsequent releases
- Manage the automated update process

If you do not have a method of installing tools and applying fix packs already working in-house or are interested in the idiosyncrasies of applying your in-house process to Rational Developer, contact your IBM marketing representative.

Deliverables

The integration analysis phase produces the following deliverables:

- An Integration Assessment report
- The Integration Implementation plan, which is input to the next phase of project: Installation, customization, and extension

- A Workspace Backup and Recovery plan
- A plan that (at least at a high level) describes the process that you use to automate client software installation and update
- Modifications to the entire implementation plan based on a more in-depth and clearer understanding of the course and roadblocks caused by software integration issues

2.4.5 Product installation, configuration, and extension

Product installation, configuration, and extensions describe the process of taking the Integration Assessment report and performing these tasks:

- Loading the parts of the optimized life cycle tool suite onto your environment
- Configuring and optimizing the installation
- Looking at areas that must be customized or extended from the product's default functionality to achieve your operational goals

We describe this phase at an extremely high level, because products and their installation processes change from release to release and the details of product installation are the subject of the product library documentation.

Role of the project manager

As project manager, there are a few things you might be able to use when your shop starts installing the tools in the optimized life cycle:

- It is important to make the distinction that installation, configuration, and for the most part, extension or customization are one-time tasks and activities. We strongly advise that you seek consultative (remote or on-site) assistance from IBM before beginning this project phase.
- Many of the technical stakeholders (your company's systems programmers, security administrators, database administrators (DBAs), and so on) that you enlisted in the implementation project team during the business justification project phase will be called on for their expertise, knowledge, and especially their authorization to run jobs that update secure regions of your mainframe (LINKPACK area, UNIX System Services, the RACF tables, and so forth).
- This phase is early in the critical path. See the previous point about ensuring that your technical staff is up to speed about what installation entails and are available to work with IBM on this project phase.
- Almost anything is technically possible with the Rational Developer and z/OS configuration. There are numerous custom exits that are written into the code and documented in the Help systems. Shops worldwide have created sophisticated and highly customized workbench shells and workflow models for Rational Developer. If you can imagine it, you can probably do it.

- However, during this implementation phase and before your pilot project, it makes sense not to try to create absolutely perfect installations and customized configurations.

The scope of the customization phase

A key element in this phase is scoping the customization of your installation. The tool suite of the optimized life cycle is comprehensive and established. There are literally hundreds of worldwide companies in the Global 2000 using these products daily. No two Rational Developer implementations are alike. Shops have customized a lot, a little, and everything in between to use these products to the maximum extent possible.

Customization typically involves writing code in COBOL, REXX, Java, or all three languages and can be simple, complex, fast, or time-consuming depending on what you are trying to achieve.

A best practice at this phase in the implementation process is to determine the scope of the customization and to scale it down to only what is necessary to prove or validate your success criteria from the pilot project. Unless you have the luxury of a lengthy pilot project cycle, we suggest that you project manage the customization to a minimum to continue down the critical path (knowledge transfer → installation → pilot project).

After the management review completes, you can resume work to customize the product suite, treating this work as any other development project (costing it out, specifying business drivers, versioning it, and so forth).

Optimizing performance with Workload Manager

Even though it is not really an aspect of customization specifically, most shops want to make Rational Developer's z/OS processing as efficient and as fast (upload/download speed) as possible.

The Rational Developer development team has created an excellent deep-dive book about the configuration of the z/OS Workload Manager and the settings that help your systems programming staff configure both the Rational Developer host components and WLM for optimized performance. Obtain *RDz & Workload Manager PDF - for optimizing RDz z/OS Performance* by Jon Sayles at this website:

http://www-949.ibm.com/software/rational/cafe/message/12915

Managing multiple configurations: Host Configuration Utility

Detailed information is available in the *Rational Developer for System z Host Configuration Utility Guide*, C14-7282.

The Rational Developer for System z Host Configuration Utility assists clients with Developer for System z host installation customization, installation, verification, and debug reporting.

The utility manages multiple configurations of a single Rational Developer for System z service level, hence providing configurations for test and production implementations of the product.

The utility helps to ease the complexity of the installation and customization of required tasks and selected common optional tasks. You can pair this function with a detailed, logged customization workflow that can be interrupted and restarted at will.

Multiple users can use the same set of configuration files, but not simultaneously. This capability allows one person to create a configuration, and someone else with the proper authority can execute specific steps of the configuration.

Stakeholders

The stakeholders in the installation, configuration, and customization process include the following roles:

- Your tools and implementation technical team: Participates in any customizations to Rational Developer that are necessary to implement your integration design
- Systems programmer, DBA, security, UNIX System Services, and TCP/IP systems staff: Have the necessary level of authorization privileges to run the mainframe setup
- Your desktop computer IT staff: Works with IBM to piece together the workstation rollout process for your pilot team
- IBM pre-sales technical specialists, IBM technical support, or a certified Rational Developer IBM Business Partner: Assist with various phases of the product installation, when asked

Process results

This process results in the following components:

- Installed, functioning, and optimized set of mainframe components
- Installed, functioning, and optimized workstation
- Customized workspace with all aspects of development administrative tasks (setting property groups, preferences, and so forth) that have been built to your shop's standards and are ready to roll out to developer desktop computers

2.4.6 Pilot project

After you have finished the product installation, you are ready to perform your pilot project. A pilot project or proof of concept (PoC) typically consists of one or more nontrivial use cases that exercise the areas of the tools from which your business justification plan seeks value. These areas often include enterprise modernization or z/OS application maintenance.

At the detailed level, there are as many separate pilot projects as there have been clients evaluating Rational Developer over the years that it has been on the market. But, most pilot projects have these key commonalities and success factors:

- Formal project management process
- Dedicated and trained team of developers
- Representative work in a limited project scope

Formal project management process

As in any project, you need the following components for your pilot project:

- Standard project plan
- Methodology for the development
- Formal start date and end date
- Project manager:
  - The project manager needs to closely manage and orchestrate the many external events between you and IBM or your IBM Business Partner and internal events among your organization and project team, meetings, documents, classes, demonstrations, installation tasks, customization tasks, testing, mentoring sessions, calls to IBM, and so forth.
  - Ideally, someone in your shop needs to be the project manager. Depending on the nature of your account, it is possible that the IBM technical sales team can assist in performing many of these tasks or at least become a single point-of-contact during your project for all pilot-related activities. Perhaps, the IBM team can own the majority of the events on the critical path. Do not hesitate to ask the IBM team for help during the pilot project and to help manage the pilot project.
- Formally stated, defined project goals, expectations, and outcomes:
  - Identify all defined milestones and results in objective, observable, and measurable language.

Dedicated and trained team of developers

Training is extremely important in ensuring that your developers understand how to use the tools and that they are able to distinguish actual product limitations from the gaps in their knowledge and skills.

Remember that pilot projects are real projects, not casual exercises. Try to ensure that the pilot team members' time is not preempted by other non-pilot project activities and that the pilot team members are able to focus on their goals and objectives with as few distractions as possible.

Representative work in a limited project scope

Your ability to extrapolate information from the results of the pilot project correlates strongly with the connection between the work your developers perform daily and the pilot project's specifications.

Choosing a representative set of tasks allows you to make determinations with confidence, knowing that the tools will directly benefit your team. If your tasks are only peripherally related to the everyday work, you only have a limited degree of certainty about your data points.

Avoid drifting by limiting the pilot project scope

Considering the harsh business timetables of today's IT software world, it makes sense and is practical to focus on the key value propositions that your company hopes to gain from Rational Developer. Do not try to accomplish too much in the pilot project. If you try to do too much, you might end up missing the things that you like and the things that you dislike.

So, achieving a balance in the pilot project scope between creating a trivial exercise and working with a highly complex application is extremely important. Based on our experience, most typical pilot projects have these characteristics:

- Involve between two and five developers that actually perform the pilot project work
- Last from start to finish from several days to at most a few weeks
- Use your application code and your application data, which differs significantly from the knowledge transfer and Proof of Technology, which typically use IBM applications and data
- Are likely to be an actual production specification, although we have seen several pilot projects that look more like academic exercises. The trade-off is straightforward:
  - Because you are comparing the experience and measuring the results of this Rational Developer pilot project, if the production specification is unique (as many are), it can be difficult to assess qualitative differences

unless you also go to the trouble of assembling a team to do the same specification using ISPF to ensure that a comparison of like activities has been accomplished (extremely difficult).

   – A project that is accomplished using traditional green-screen technology by the same group can give you a baseline for a good, bad, or indifferent comparison.

- Focus on one product, not the entire optimized tool suite. There are tremendous advantages to the built-in integration among Rational Team Concert for System z, Rational Developer, and IBM Rational Asset Analyzer, the problem determination tool, CICS Explorer, IMS plug-ins, Universal Driver, and so forth

The pivotal position of management during the pilot project

By now, we hope that the central theme of having a strong project manager is clear. We define "capable project management" as a pilot project that nets you the facts and results that you need to make a management implementation plan based on empirical data points.

Without a strong, proactive project management position taken (possibly shared between an IBM counterpart or you), you run the risk of not achieving your goal to fairly and impartially evaluate the tools of the optimized life cycle for your company.

Measuring results

During the pilot project, teams typically focus exclusively on achieving the goals that are documented in the specifications. So, distractions, such as daily evaluation surveys, questionnaires, and so forth, are typically an unwelcome intrusion, although journaling (and detailing) any exceptional experience (good or bad) is typical.

The primary measurement of any pilot project is, of course, using the tools of the optimized life cycle. Ideally, the developers are able to finish the pilot project on time and within the parameters that are set forth in the specifications.

Next, you determine if the new technology met your expectations:

- Did the new technology measure up (or exceed expectations) in quantified terms?
- In what ways did the new technology not measure up to your expectations?

Typically, you also ask these subjective (but important) questions:

- What was each developer's experience using the new tools? This answer is often expressed as a quantitative opinion on a scale from 1 to 5.

- What were the strengths and weaknesses of the tools? You might ask the developers open-ended or multiple choice questions.

Populating a spreadsheet that documents the evaluation parameters of the pilot project alongside statistical summaries of developer reactions is standard. Including several or many of the answers to open-ended survey questions often provides a qualitative edge to an admittedly imperfect science of evaluating something as new (to many) as the tools of the optimized life cycle.

Group consensus meetings

Often, it is customary at the end of the pilot project for the entire team to come together, discuss the implementation project, and seek agreement in a parliamentary process. These group meetings (sometimes referred to as consensus-building sessions) have their good and bad points:

- On the plus side, meeting together can provide a concordant view of results, making communication to management simpler and making individuals more secure in their majority rule verdict. Also, getting everyone together for an hour is probably less expensive than meeting with each person for thirty minutes.
- The peer pressure that is exerted by group dynamics can be a big negative. If a perception exists that management wants or does not want the tools to be used, this perception can affect what is said in a public forum. Or, a senior technical person can often influence individual opinions, thus, stifling free expression.

A private, anonymous survey or questionnaire that quantitatively measures the results of the pilot project is a better way of retrieving information and opinions, such as *"Rank on a scale from 1 (negative) to 5 (positive) the following topics."*

Measuring MIPS reduction and Rational Developer's effect on performance

Successfully finishing a pilot project and quantifying results are usually all the validation companies seek to provide balanced factual input to the Management Review and ultimately to the implementation plan.

But, occasionally, the team wants to revisit the business case and provide additional justification if the pilot project was a success. The team can provide financial justification (pricing from IBM). And, there are really only two measurements and inputs of any significance to management: millions of instructions per second (MIPS) savings and productivity improvements.

Measuring MIPS reduction is the more difficult of the two measurements to perform. You need to perform these tasks:

- Set up a “clean” or isolated logical partition (LPAR)
- Write detailed scripts
- Create an empirical testing methodology or process
- Actively involve the systems programming staff in iterative test runs
- Run System Management Facilities (SMF)/Resource Measurement Facility (IBM RMF™) reports

Of course, ask for additional time from the development staff. Setting up the environment (a clean LPAR) and configuring and running SMF/RMF reports are not trivial exercises, and the time demands on your systems programming staff can be considerable. Measuring productivity is generally the simpler of the two optional pilot project phases.

Measuring productivity

It is not easy to measure productivity. A lot of anecdotal evidence swirls around every new product on the market, but most software practitioners are skeptical of productivity numbers published by either vendors or users of vendor products. Productivity is an extremely elusive property to measure empirically, because it depends on a large number of independent variables, including but not limited to bias, participant experience and skill level, and so forth. Clients always seek additional evidence or confirmation of improved productivity when purchasing software.

IBM 2010 IDE efficiency study

In an attempt to look at two IBM products (to remove vendor bias), IBM performed benchmarks comparing ISPF V6.0 (latest release) and IBM Rational Developer V7.6.0 in first quarter 2010. The research consisted of 23 developers that performed two detailed task lists, which were composed of 100 discrete operations that comprised standard or typical z/OS maintenance and development work and were instrumented at a low level of detail, such as move your mouse over field X, press F3, press the Page Up key, and so forth. The developers performed one list of development operations using ISPF on an extremely fast IBM z10™ (subsecond response time for all activities, including JCL submits). The developers performed the other list using Rational Developer V7.6 on Lenovo Thinkpad personal computers (a mix of T60p and T40 models).

The scripts covered the following daily development tasks, based on an ISPF-style workflow:

1. Source navigation
2. Source editing (ISPF edit operations)
3. COBOL statement coding

4. Syntax error removal
5. Build operations (Compile/Link)
6. Data flow analysis
7. DB2/SQL work (test table data manipulation and SQL statement development and testing)

IBM published the results at this website:

http://www-949.ibm.com/software/rational/cafe/thread/4056

The results show that, when performed by a group of veteran TSO developers, tasks accomplished using Rational Developer were completed in an average of 30 percent less time. The website also contains attachments for the actual scripts, which, because of the precise level of detail, allow for a true comparison between the products. IBM added additional controls into the research, and a PDF, which is also available at this website, describes these controls.

We advise that if you are considering adding on to the pilot phase of your implementation to research the potential productivity benefits of using the tools of the optimized life cycle, pattern your work on the comparison model evidenced by IBM in its research.

Stakeholders

The following individuals are the stakeholders in the pilot project phase:

- Your development staff: Use the tools of the optimized life cycle integrated into your existing z/OS infrastructure to deliver measurable functionality
- Your tools team, systems programmers, DBAs, security, TCP/IP, and desktop computer IT systems staff: Support the development staff on an as-needed basis, exactly as is currently the situation with your staff and systems today, but with any work they have accomplished installing, configuring, and customizing the tools for this pilot project
- IBM pre-sales technical specialists, IBM technical support, or a certified Rational Developer IBM Business Partner: Assist your development staff with either on-site or remote assistance, answer questions, field “how to” questions, and mentor beyond the initial education that is received during the knowledge transfer phase
- Project manager: Possibly partners with IBM to orchestrate the events and work that surround the actual pilot project team and their tasks

Result and deliverable

After this phase, you have the following result and deliverable:

- The functionality that was specified in the pilot project plan now completely available

- Documentation that describes the process of performing the pilot project, including an evaluation of the pilot project and input into the management review phase

2.4.7 Management review

When all the pilot project reviews are complete, you summarize the quantitative survey results and present the findings to management. The corporate sponsors weigh all the factors that the pilot project uncovered and create the implementation and rollout plan for the tools of the optimized life cycle.

Usually, on the way to this step, you communicate your findings to IBM and allow IBM and its development staff to respond to any issues prior to the implementation and rollout plan.

There are several reasons to discuss your findings with IBM first:

- Occasionally, the client sees a function or feature as mandatory or crucial to the product's success and that function or feature is not currently in the installed version. IBM is extremely careful about not pre-announcing future features and functions; however, if your company signs a nondisclosure, it might be possible to get information about specific features or functions that you deem critical.
- Occasionally, a technical hurdle emerges during your pilot project for which the knowledge transfer phase, which occurred much earlier in this process, did not prepare you. There can be many reasons for this to happen, such as the development team not covering what they learned in class or the team forgetting how to do something. But mostly, a technical hurdle is simply due to the fact that every mainframe shop differs from every other mainframe shop in the world. Our experience is that no two development organizations have the same products, process, or mix of methodology, tools, and customizations to the IBM tools.

These obstacles might be standard integration challenges, but it is better to work through the challenges with IBM. Therefore, you need to be in close contact with IBM before and during the management review.

Hopefully, if you managed the pilot project closely, you identified, documented, communicated and responded to most if not all must-haves during the pilot project. However, occasionally, these engagements are extremely short and time does not permit you to identify everything you need.

Given the amount of time and effort invested by both parties, it is appropriate to open a channel for discussing and addressing any last-minute concerns (if any concerns exist).

Assuming that all the evaluation criteria have been met by the pilot process team and that a financial case has been made for rolling out Rational Developer in a controlled fashion to the field, only three steps remain in the process:

- Human factors analysis
- Rollout
- Self-sufficiency

2.4.8 Knowledge transfer (for the product rollout)

We have discussed the knowledge transfer needs for the teams and individuals performing the integration analysis, installation, and customization, and the pilot project. Now, we discuss your platform migration (from ISPF to Rational Developer) education for the rollout.

There are two components that go into your analysis:

- Knowledge transfer, which is now expanded to encompass requirements for rolling out to the broader dimensions of your development teams and not merely specific to the pilot project
- Customization of the default workspace

The information presented in this section is not in-depth. We describe the lessons that we have learned in product education, especially paradigm-shifting education, to help individuals who have worked for 20, 30, and more years in a green-screen environment to move to the Eclipse and Rational Developer graphical user interface (GUI) platform.

Customizing the knowledge transfer

Immediately after developers have completed training (right after, not weeks after, because the human propensity to forget learned behaviors over time is well documented), you must expose the developers, who comprise the pilot project team, to the tools through production-quality work, with your code and your data during the pilot project. As part of the evaluation process, look carefully at your development community, potentially breaking it out into the following three groups:

- Entry-level developers (group 1)
- Developers with fewer than 20 years of TSO development experience (group 2)
- Veteran TSO developers with 20 and more years of experience (group 3)

Group 1 learning patterns

Why this categorization? The first two groups, in our experience, take to Rational Developer and its Eclipse workbench organization and development paradigm quickly and easily, especially the first group, who will most likely embrace Rational Developer quickly. Entry-level developers require little to no convincing that Rational Developer is a much better tool for performing their work, because they might have used Eclipse in college and certainly are intuitive mouse/GUI-based developers.

Training needs for this group

The first group needs product education, but they need education based on an Internet-search style of education or a self-reliant "I'll figure this out one way or another without hassling anyone" approach to acquiring skills that got them through college.

Entry-level developers have typically developed skills that allow them to get answers regarding new technologies and especially GUI-based products quickly and intuitively. They almost never require follow-on mentoring. They need to follow an in-depth product education curriculum, but in our experience, they take the training and become adept at the new tools.

Entry-level developers are excellent candidates for either self-paced or distance learning education, because they are self-reliant learners, who are not intimidated by the Eclipse interface and usage model. For example, they figure out that an option can be found under the Preferences page, because they have spent years working with similar technology.

Internal mentors and help center staff

You can count on this group to provide mentoring to the other two groups. In fact, many shops staff their Rational Developer Help Center with group 1 developers, who have picked up on the new tools quickly and thoroughly. Most likely, many developers in group 2 and most of the group 3 developers benefit from mentoring to ramp up quickly. In our experience, the group 1 developers are a natural fit for internal product mentoring.

Remember, group 1 knows little about your z/technology stack and less about your applications and business. However, group 1 can rapidly learn the tools of the optimized life cycle and contribute to your rollout.

Group 2 learning patterns

Group 2 is also likely not averse to mouse/GUI (Microsoft® Windows® style) development, but, it is unlikely that these developers have used Eclipse tools (unless they know Java or Java 2 Platform, Enterprise Edition (J2EE).

However, group 2 typically has had experience with Windows, not DOS or character-based development tools, and can transition to Rational Developer/Eclipse easily. Although, as experienced TSO developers, they want to use Rational Developer in their day-to-day production work.

Often group 2 developers make excellent pilot team members, because they are unbiased and open to the paradigm shift from green-screen to GUI. They are also experts in z/OS development and understand the level of sophisticated product feature support that is necessary to do their jobs end-to-end.

Training needs for this group

Group 2 requires a complete training curriculum, seeing all phases of the product and understanding the analogies with their ISPF tools. Group 2 with its in-depth expertise can be Rational Developer thought leaders internally, because often these individuals are lead technicians, looked to internally for solving technical issues. It is critically important to gain their agreement, but it requires an in-depth education curriculum, with qualified instructors willing to answer questions outside of the materials that pertain to both deep-dive z/OS topics and your shop's software development practices.

Instructor-led training or distance learning

We advise that group 2 learn in an instructor-led educational setting; however, we have found that distance learning can work with these developers.

Often, there are early adopter types in this group, developers that have attempted to remain contemporary and are eager to learn and use new tools. This group is really a more seasoned group 1, when it comes to knowledge transfer. These developers are good candidates for becoming internal Help Center guides, mentors, and technical champions, because they own both the old and the new technologies at your shop and they are often established technical experts.

Group 3 learning patterns

Group 3, which includes veteran TSO developers, is the challenge. Group 3 developers have these characteristics:

- Are typically closer to retirement and, thus, not motivated to learn a new development IDE
- Have invested over 20 years in ISPF, which has evolved into reflexive and almost involuntary ISPF development techniques, with ingrained keyboard skills. Adapting to a new IDE, even one with an ISPF profile, can be challenging.
- Are not fluent with a mouse

- Are often the go-to technical and business experts in a shop, because they have an enormous experience base of combined business and application technical knowledge and therefore represent a critical constituency and challenge.

Training needs for this group

It is our experience that with group 3, you must always educate them starting from a TSO point of view, presenting topics with ISPF analogies, pointing out that there is almost a one-for-one equivalent technique from ISPF to Rational Developer and that they have the ability to learn this new IDE.

You can look at an example of this approach in the Rational Developer Workbench - Introduction for ISPF Developers course at this website:

http://www-949.ibm.com/software/rational/cafe/docs/DOC-3129

The value of instructor-led training

Note that the group 3 TSO developers benefit the most from live, instructor-led classroom training. Because the richness of the Eclipse shell might intimidate these developers, the initial impressions are extremely important. It is critically important that you have an instructor in the room to clear up misunderstandings and facilitate the learning in real time for group 3 developers.

The value of instructor-led training is equal to the teacher's work experience performing z/OS development. Not all product features are equal. Only someone who has actually performed maintenance and support work can emphasize those tools and options that are essential to learn versus other features that are merely nice to know.

Training options: Where and how to train for the rollout

You must decide where and how to train your development staff. Similar to 2.4.3, "Pilot project installation and configuration knowledge transfer" on page 37, you have the following options:

- IBM Training and Education Services
- A Certified IBM Business Partner
- Your internal training staff
- Another option, such as working with the IBM pre-sales team (contact your account team for more information)

The advantages and disadvantages include the cost and value of customization and vendor-led training expenses (per event charges) versus in-house training expenses. If you have questions in this area, talk with your IBM account team.

Customizing training materials, labs, and workshops

One key to success in quickly attaining the goals of your product rollout is the seamless integration of your staff's new skills into your production development practices. Every shop is unique with respect to its product mix and process approach. If your training goal is to transfer technology skills to your entire development base, you must customize the training to include your process and methodology, especially as it pertains to your SCM custom configuration and the use of any additional custom development tools that you think are appropriate.

You must answer these questions:

- Are these tools integrated into the new IDE?
  - If so, how do you access these tools and what has changed in the process?
  - If not, how do you access the tools on the host from Rational Developer?
- What development tools do the new products replace? Which development tools are altered and which development tools are complemented?
  - A development architecture reference document is an excellent way of communicating the changes. See Figure 2-3 and Figure 2-4 for examples.

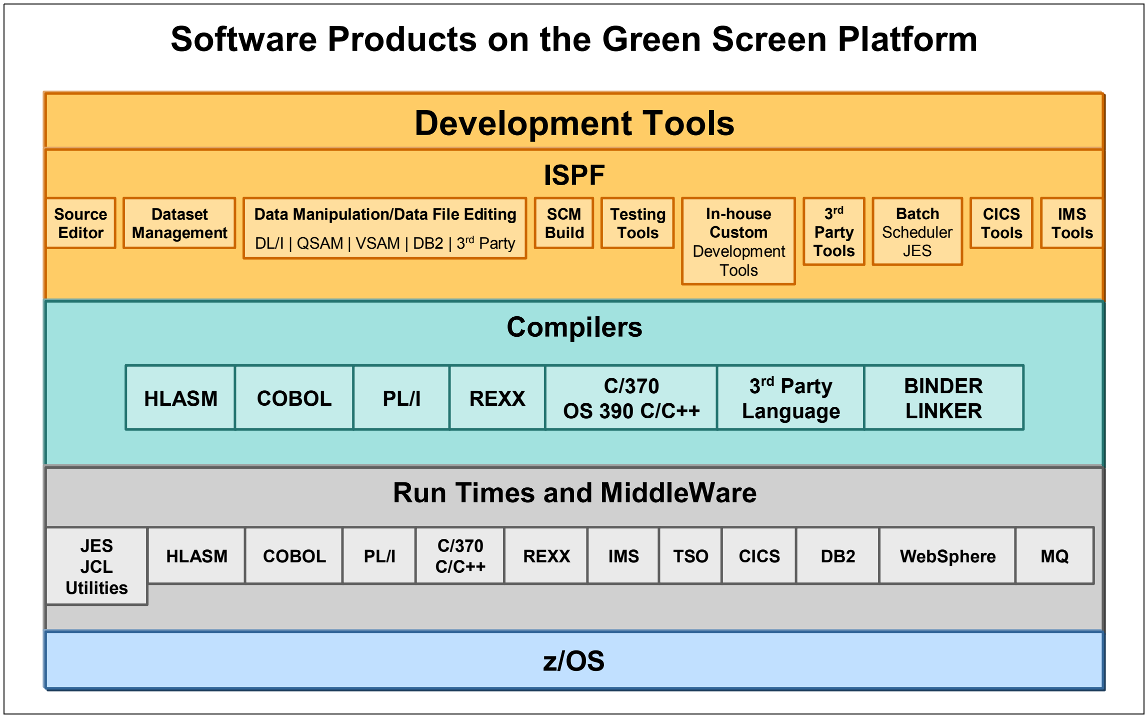


Figure 2-3 - Reference Architecture - Green Screen

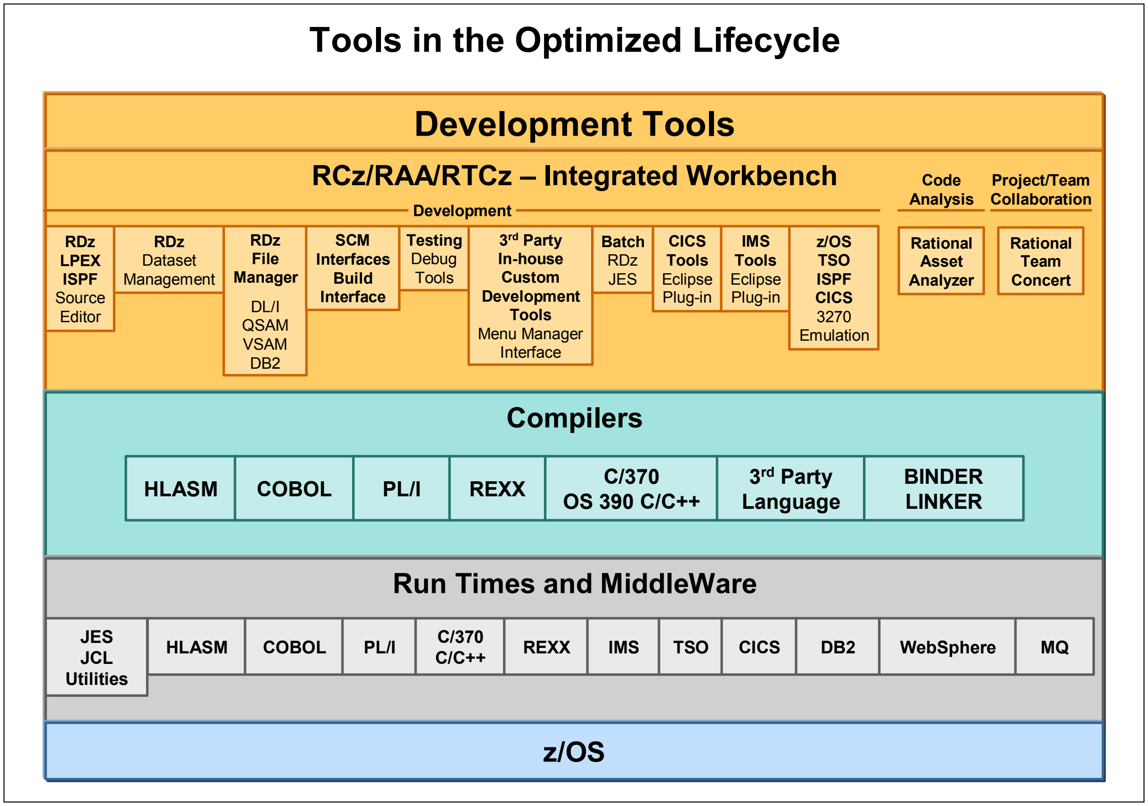


Figure 2-4 - Reference architecture in the optimized life cycle tools

Achieving training goals

Achieving these goals requires you to create your own custom training materials and to maintain them. Most IT shops do not have the resources and extra capacity to create their own custom training materials. Consider these sources for training materials:

- IBM Rational Education Services has well-crafted materials that you can purchase and customize to your shop's specifics, and they have access to educational content specialists who can help you customize the materials.
- IBM Business Partners, who also have expertise and experience in this area, offer worldwide Rational Developer consulting and education services:
  - Ask about your IBM Business Partner's Rational Developer qualifications; in particular, see if your IBM Business Partner is certified:

    http://www-03.ibm.com/certify/certs/38005001.shtml

- You also might be able to get educational materials from the IBM pre-sales team that performs Proof of Technology engagements and other mentoring during pilot projects. Ask about the availability of learning artifacts that you can use in-house to jump-start your own training process.

The importance of customized workshops

If creating and maintaining custom Rational Developer course materials (lecture and demonstration slides) are outside your area of responsibility or domain or if creating and maintaining custom Rational Developer course materials is an aspect of product implementation for which you simply did not budget, at least, customize the labs and workshops that your developers work through to master the new tools.

In our experience, performing hands-on labs is where 90+ percent of the learning and assimilating occurs (as opposed to watching a demonstration or reading PowerPoint slides). So, if time and money are concerns and if you have to focus on one area of knowledge transfer to accomplish, customize and perfect the workshops.

Consider these options to customize and perfect the workshops:

- Adding on to the existing labs in the courseware by (commonly used approach) reworking the instructions to point to your own programs, libraries, job control language (JCL), basic mapping support (BMS), and so forth:
  - The benefit of reusing content is obvious (faster time to complete the development).
  - The drawback of reusing content is that possibly nuances exist in the labs so that finding your own resources to substitute might not be as easy as it seems
- Working with IBM Educational Services or IBM Business Partners to create custom workshops

Customizing the default workspace

As part of your knowledge transfer analysis, you might need to set up two default workspaces to roll out:

- An Eclipse-style workspace that uses most of the default settings, which fits the development style of group 1 developers and the early adopter developers in group 2
- A workspace (see Chapter 3, “Eclipse: The optimized life cycle integrated development environment” on page 67 for a description of the workspace) that provides a smoother training experience for the rest of group 2 and the group 3 TSO veteran developers

Two separate workspaces are not always necessary. Creating, testing, and maintaining two workspaces for developers require additional work. But, because one of the benefits and purposes of a workspace is to provide a customized environment, and because group 1 and group 3 differ dramatically in their needs, you can resolve usage model and human factor issues by setting up separate workspaces.

Summary

Too many times, we have seen the hard work of the client's staff, pilot project team, tools team, systems team (systems programmers, DBAs, IT, and so forth), the project manager, and the executive, put into rollouts, yet be negated by too little attention paid to educating the development staff.

We have shown you the options, trade-offs, benefits, and costs. You must train your staff on the tools of the optimized life cycle in preparation for rolling out the tools. The standard Rational Developer curriculum is between two and three days in length, which is a small price to pay for a product that delivers the kinds of productivity benefits and cost reductions that are available with Rational Developer.

Stakeholders

The human factors analysis phase includes the following stakeholders:

- The implementation team: Performs the analysis on your development staff and prioritizes work to be performed in the area of customizing your rollout deliverables
- Your tools team: Creates the customized workspaces, including the build files and so forth
- IBM specialists or IBM Business Partner team: Assists either with the training, actually performs the training, or provides the raw materials for you to customize and conduct your own in-house training
- Project manager: Possibly partners with IBM to find the right individuals, companies, resources, and collateral

Deliverable

This phase produces a training plan on which corporate purchasing can act, including these aspects:

- Justification for training
- Approach to training
- Level of customization
- Contacts and options for training

2.4.9 Rollout

Before rollout, you have completed all these steps in the implementation plan:

- Assessing the integration points and creating the necessary integration software junctures
- Analyzing the knowledge transfer requirements and building a training and mentoring program that is customized to your installation specifics (taking into consideration the unique learning requirements for all the developer groups)
- Installing and optimizing the mainframe components
- Creating a model workspace for propagating across your organization
- Managing backup and recovery of the desktop and developer workspaces

The rollout process

The rollout consists of this process:

- Communicating management imperative messages:
  - Conveying to the development community and especially to project managers the business and technical goals that the tools of the optimized life cycle will help you reach
  - What these goals mean to the company
  - The means of accomplishing the goals using the new products
- Scheduling the desktop installations:
  - Rational Developer for System z client software
  - Default workspace
- Verifying that the desktop installation is in place and fully functioning:
  - Hopefully, you can run a semi-automated installation verification process or at least have individuals execute a script to make this task less labor intensive
- Following up on additional integration opportunities:
  - Using IBM Rational Host Access Transformation Services for integration with ISPF dialog panels
  - Integrating with your SCM (additional integration is usually necessary)
  - Integrating more deeply into the Rational Developer/Eclipse framework (requires Java/Eclipse plug-in coding)
  - Integrating additional REXX/CLIST scripts through Menu Manager

- Identifying the individuals or teams that will use the tools:
  - Ensuring that these developers work with Rational Developer
  - Using Rational Developer on actual z/OS projects within a few days after completing the class
- Scheduling training:
  - Remember the critical importance of just-in-time Rational Developer product training. Schedule training classes for new Rational Developer developers within a week (two weeks at most) of their first Rational Developer project to maximize learning and retention.
- Providing the team members (especially group 3 developers) with a mentor or an internal and external Help Desk process
- Following up:
  - Weekly checkup sessions for one month
  - Monthly checkup sessions for 1/2 year

The significance of the management imperative

We have seen the vital importance of corporate management supporting new initiatives to promote the use and adoption of new technologies.

In certain organizations, management directives can help advance activities. In other organizations without management imperatives, perfectly attenuated goals can disappear or never be reached, which is not in anyone's best interest.

So, it is important that management clearly communicate the reason for adoption of the new tools and really support the rollout. This management endorsement might seem obvious to you, but surprisingly, it does not always happen.

There is limited IBM involvement in the rollout. We advise that you continue working with your account team to periodically communicate your progress, exchange ideas, and discuss and follow up on any issues.

2.4.10 Becoming self-sufficient

As your shop matures using the tools of the optimized life cycle, we have seen that shops typically begin to establish their own internal center of excellence or Rational Developer Help Desk. We have seen shops create their own internal user group that focuses on sharing experiences, communicating internal procedures and changes (perhaps to your product customization), and, in general, extracting more value from Rational Developer.

On your path to self-sufficiency, you might choose to take these steps:

- Move from using the IBM learning materials to your own lecture content (we have already discussed the key value and importance of integrating your own workshops).
- Create your own shop-specific plug-ins.
- Contribute to the IBM Rational Developer user group meetings as product technical experts.
- Publish articles or write blogs about Rational Developer.
- Integrate your own product documentation into the Eclipse shell.

2.5 What you can expect from IBM

IBM has a strong and abiding interest in your success with our technology across your entire company, not just success with the tools of the optimized life cycle.

You likely already use a great many IBM products. Replacing one product with another, even if you replace a long-serving older product, is not something that you (and we as your production development, compiler, and runtime software provider) take lightly.

IBM can provide you with several fee and no-charge resources that can assist with all aspects of impartially evaluating and then (when you decide to move forward) implementing the tools.

IBM offers these resources:

- Implementation project plans
- Meetings to discuss the business case for the tools, including:
  - Product demonstrations
  - Recorded demonstrations
  - Data sheets and slides
- Project management assistance, guidance, support, and even role-sharing
- Impartial development architecture health assessment consultation and study
- Product installation documentation and on-site and remote assistance
- Product customization, including integration with your SCM documentation and on-site and remote assistance. Discuss this resource with your account team, because customization is a broad term and many aspects of customization are part of installation

- Proof of Technology demonstrations and opportunities for you to test-drive products. Schedule these activities with your account team.
- Pre-sales and post-sales education and product mentoring:
  - On-site
  - Remote
  - Materials and content
- Premier technical support during your pilot project:
  - On-site
  - Remote
- Online user groups and other collaborative learning engagements

Many of these IBM resources are fee-based, but many resources (probably more than you think) are at no charge. Contact your IBM account team to discuss how to obtain and sign up for resources.

2.6 Wrap-up

Without strong and proactive project management, implementing the tools of the optimized life cycle rarely succeeds due to many factors, but primarily because every z/OS development shop is unique and requires its own customized approach to the tasks, events, and implementation plan.

Part 3

The tools in the optimized life cycle: Practitioner

This section introduces several of the software products that comprise the optimized life cycle:

- IBM Rational Team Concert for System z
- IBM Rational Developer for System z
- IBM Rational Asset Analyzer
- CICS Explorer
- IBM Problem Determination Tools
- IBM WebSphere MQ for z/OS
- IBM WebSphere ILOG JRules for z/OS

We describe these products primarily from the practitioner's or business application developer's point of view. We emphasis technical details, and we mention several how-to steps and information to document the usage models.

To understand the use case step-by-step examples in Part 4, “z/OS application maintenance and support” on page 473 and to be able to visualize the process, it helps to see screen captures of these tools and to see examples of what they can do, as well as learn about their limitations.

Project managers with a technical background can also learn the capabilities and limitations of these tools, as well as understand the tools’ technical benefits.

The information in Part 3 is only a high-level technical overview. This information does not replace the product documentation, product education, mentoring, and technical enablement services.

You can obtain more information about product education and services by visiting the IBM Rational developerWorks site:

https://www.ibm.com/developerworks/rational/

Also, visit the COBOL and RDz Hubs in the Rational Cafe:

https://www.ibm.com/developerworks/rational/community/cafe/index.html

3

Eclipse: The optimized life cycle integrated development environment

This chapter describes the Eclipse framework, tools, concepts, and vocabulary from the vantage point of a z/OS Interactive System Productivity Facility (ISPF) developer.

If you already use Eclipse at work or, perhaps, used it in college, you might not need the information in this chapter and can skip to Chapter 4, “IBM Rational Developer for System z” on page 83.

But, if you have never used Eclipse, you need this background to get the most out of the rest of Part 3, “The tools in the optimized life cycle: Practitioner” on page 65 and Part 4, “z/OS application maintenance and support” on page 473.

This chapter contains the following topics:

- Introduction to Eclipse
- Eclipse menus
- Eclipse editors
- Eclipse views and perspectives
- Eclipse and ISPF comparison
- Eclipse and property files: Customizing your development options

3.1 Introduction to Eclipse

Eclipse is a multiple language, integrated development environment (IDE) that is used worldwide for software development in modern distributed platforms and traditional z/OS platforms.

The Eclipse software is written predominantly in Java. Developers use Eclipse to develop COBOL, PL/I, IBM High Level Assembler (HLASM), Java, C/C++, PHP Hypertext Preprocessor (PHP), and many other languages.

In addition to language-based development, you can use Eclipse to build applications for z/OS that include batch (Job Control Language (JCL)-driven), Information Management System Transaction Manager (IMS TM) and Data Language/Interface (DL/I), CICS, WebSphere MQ, and other applications.

Eclipse is highly configurable and easily customized and extended by both software vendors and installations that have standardized on it. In this book, we describe Eclipse as the foundation software of the Enterprise Modernization Workbench.

As a workbench, Eclipse consists of menus, editors, views, and perspectives (Figure 3-1).

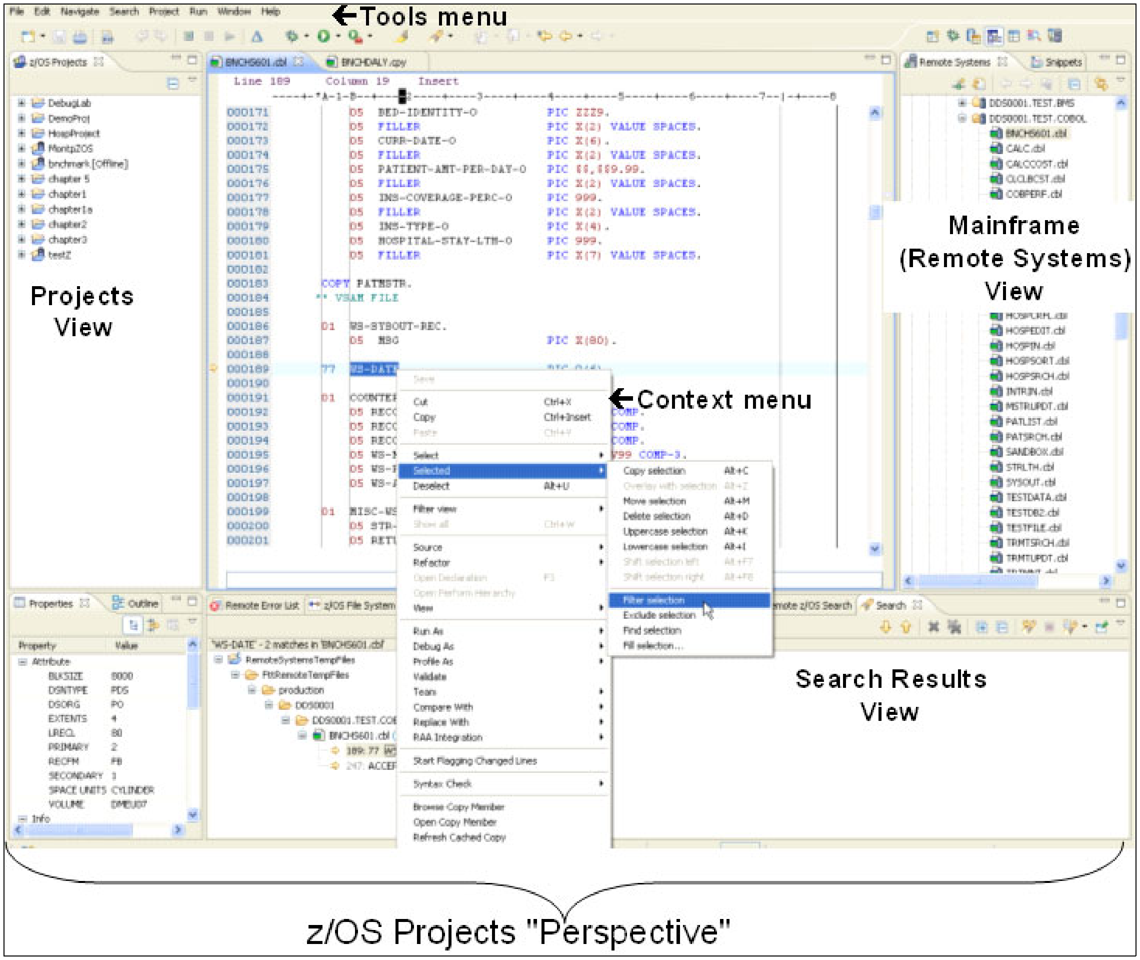


Figure 3-1 Eclipse elements: Menus, views, and perspectives

3.2 Eclipse menus

Menus have two functions in software development tools:

- Navigation: Use menus to transfer to areas within the IDE to perform another task.
- Operation: Use menus to invoke a product feature or function (Figure 3-2).

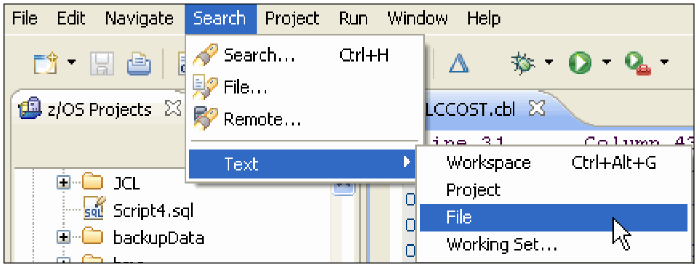


Figure 3-2 IBM Rational Developer for System z z/OS Eclipse tool bar menus

You use Interactive System Productivity Facility (ISPF) menus primarily for navigation, because the software is divided into the hierarchy of functions that make up ISPF. The IBM Rational Developer for System z menus come in two varieties:

- Tool bar menus (Figure 3-2)
- Context menus (Figure 3-3)

3.2.1 Tool bar menus

The tool bar menus are fashioned around the standard Common User Access (CUA) design:

- File menus allow you to create new resources, import, export, and so forth.
- Edit allows you to copy and paste and manipulate source code and sequential data sets (queued sequential access method (QSAM) files).
- Search allows you to find and replace text either on your workstation or in libraries and files on z/OS.

There are other product-specific menu options that we describe in other sections of this book.

3.2.2 Context menus

Most Eclipse-based products use the *context* development concept. The context method means “whatever I have selected” in the workbench. So, a context menu provides access to available functionality that allows you to do something with whatever you have selected in the workbench. This context concept is a universal GUI development standard that transforms (without context menus) a series of typing commands into one or two mouse clicks (with context menus), as shown in Figure 3-3:

- Select an item (text, a file, or a property)
- Right-click and, using the context menu, act (copy, paste, refactor, filter, and so forth)

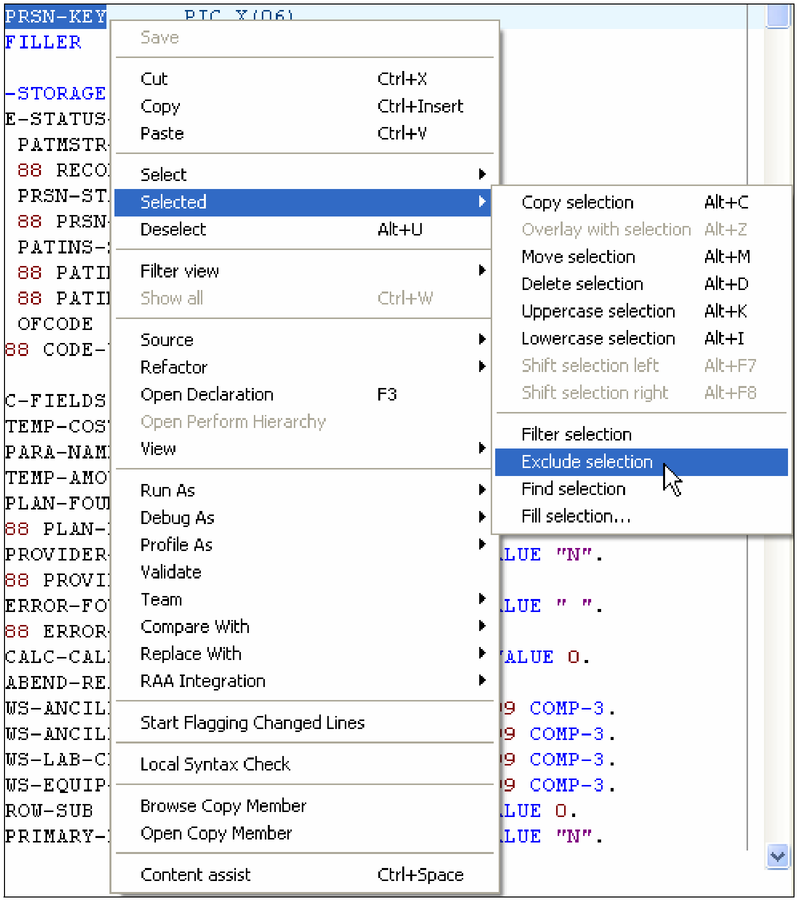


Figure 3-3 Eclipse context menu

Figure 3-4 shows the action of selecting a COBOL variable named PRSN-KEY. With this variable selected, exclude all lines of source from the program that contains that variable name. Using context menus is convenient, fast, and easy to learn when you understand the purpose and use of context menus.

3.3 Eclipse editors

Functionally, Eclipse is stronger than ISPF in the areas of language-sensitive editing and highly configurable editing. In Figure 3-4, note the following characteristics:

- Colored text elements (green comments, blue keywords, and maroon literals)
- Persistent:
  - COBOL A and B margin column indicator
  - Line and column number for the cursor
- Current line background color
- Language Intellisense (Content Assist) editing, which includes a ToolTip for the group-level structure variable and the variable’s COBOL PICTURE clause
- ISPF prefix area

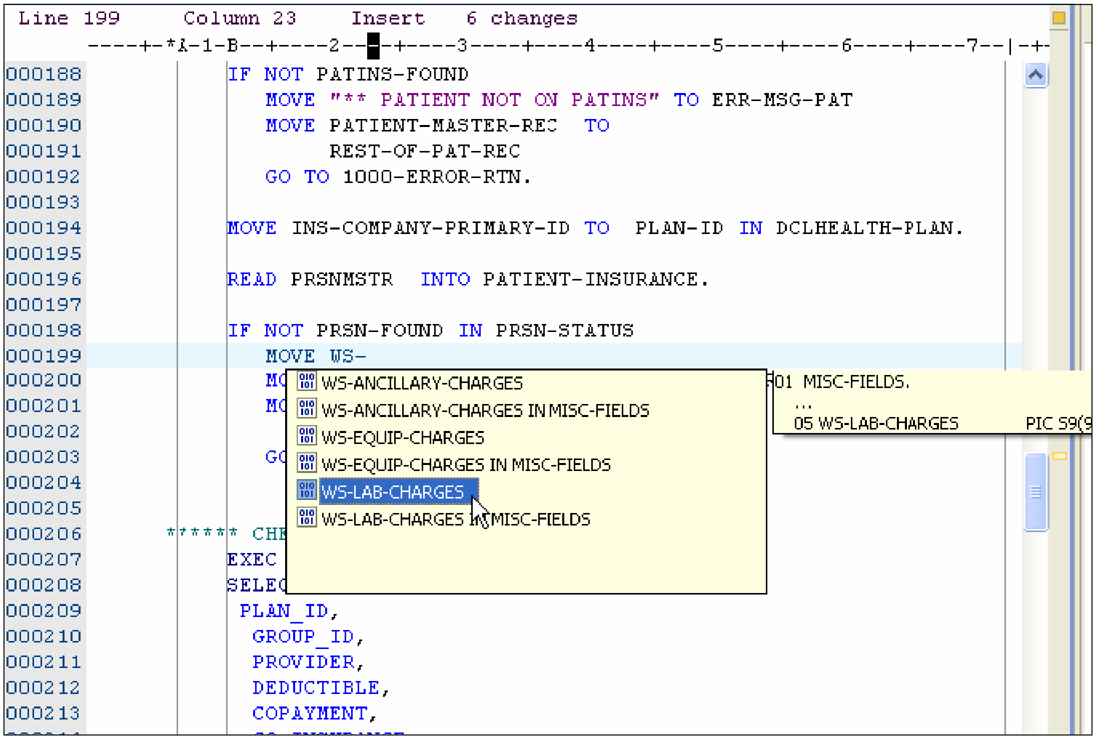


Figure 3-4 COBOL editor

Eclipse loads the correct language-sensitive editor for the type of file that you work with. Figure 3-5 and Figure 3-6 show various kinds of editors. Note that Eclipse invokes a specific editor based on the file type extension: .cbl, .cpy (for copybooks and includes), .asm, .bms, .jcl, and so forth. You can configure the file type extensions and also many other aspects of the editing operations.

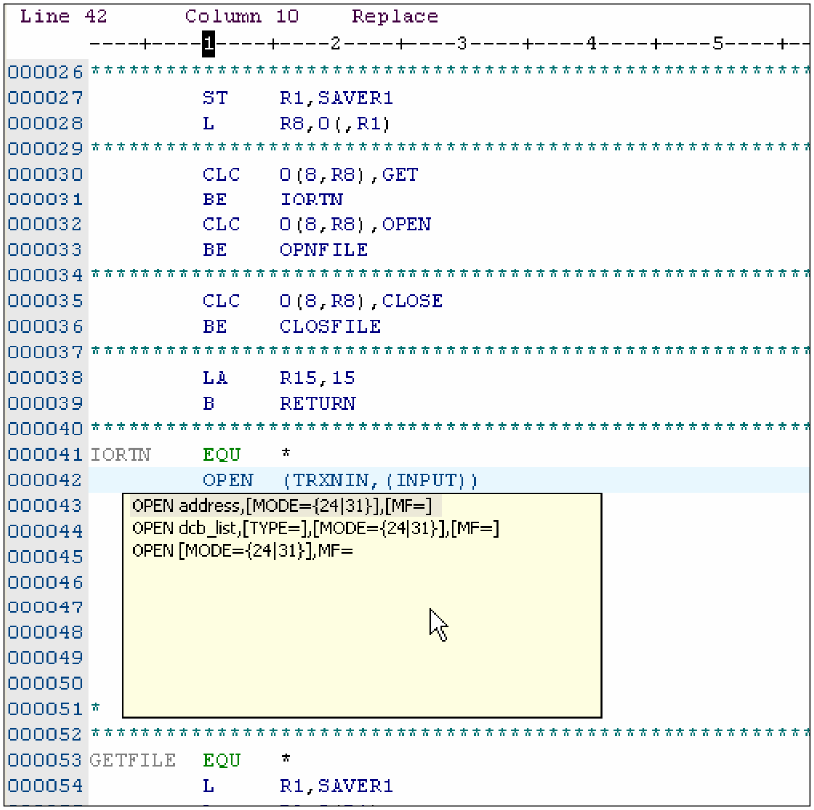


Figure 3-5 HLASM editor

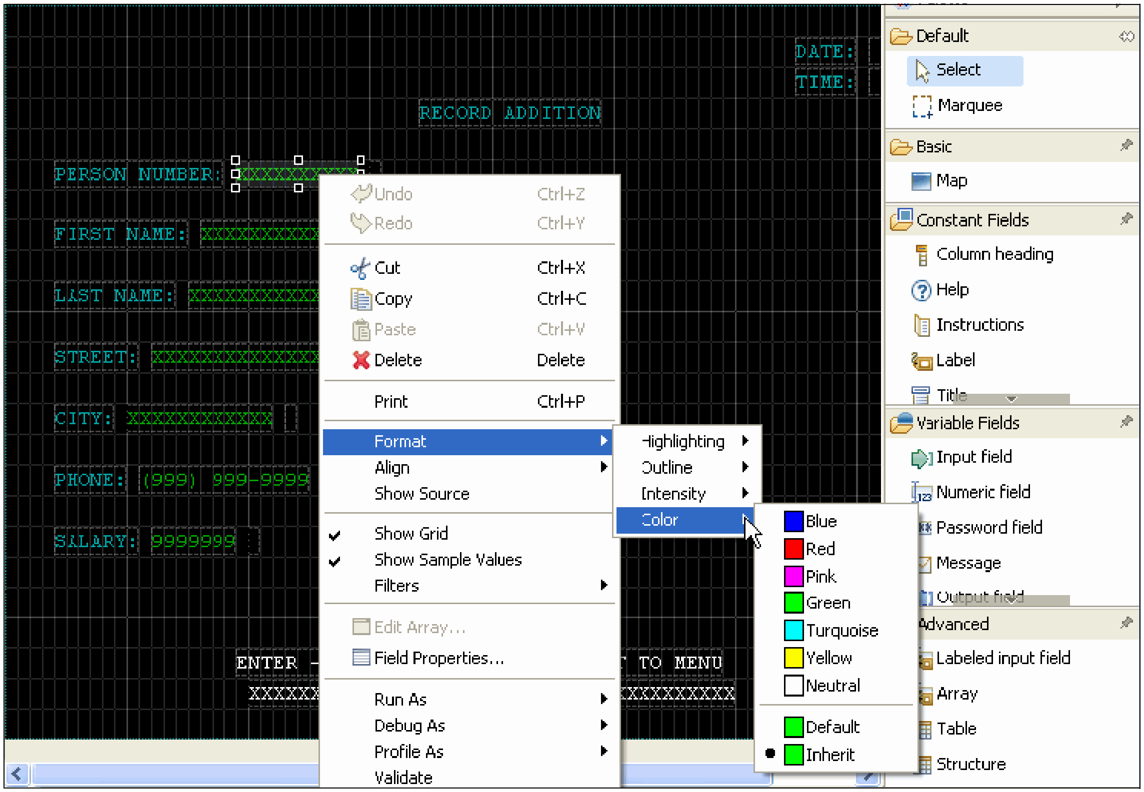


Figure 3-6 Basic mapping support (BMS) map editor

3.4 Eclipse views and perspectives

In this section we explain what Eclipse views and perspectives are.

3.4.1 Views

Figure 3-1 on page 69 shows that the Eclipse user interface consists of multiple, small, graphical windows called *views,* as opposed to ISPF, which is a character-based 3270 user interface (UI). In Eclipse, each view represents a single high-level function, for example:

- Your projects
- Your mainframe resources
- A text search result
- Source code syntax errors

These views are the ground-level building blocks of your workbench. From this foundation, you can perform these tasks:

- Add views to your workbench
- Subtract views from your workbench
- Combine and resize views quickly and easily

You can customize the views of your workbench to make your development UI optimal for a specific task. For particular tasks, you might want certain views maximized, minimized, or resized according to what you want to view. See Figure 3-7 for an example of custom views.

There are over 100 separate views in most of the Rational products, although you only work with about a dozen views when working with z/OS. We describe the relevant views in subsequent sections of this book, but for now, consider each view as the functional equivalent of a single ISPF option or panel (Figure 3-7 on page 77).

3.4.2 Perspectives

Eclipse combines views into a *perspective*. Perspectives allow developers to work with a set of views related to a given high-level task.

The following perspectives are examples:

- Data perspective: Allows you to perform DB2, SQL, and IMS database specific work, testing statements and modifying table and segment values
- z/OS projects perspective: Combines views that relate to z/OS COBOL, PL/I, and HLASM project work
- Debug perspective: Used for z/OS (remote) and local application debug
- Enterprise Service Tools perspective: Aggregates views that are specific to creating and testing Web Services

Like views, you can customize perspectives. Figure 3-1 on page 69 shows the IBM Rational Developer for System z z/OS Projects perspective with default views. Figure 3-7 shows the Debug perspective, which is enhanced with several views for a better understanding of the events and data values that are in a unit test session.

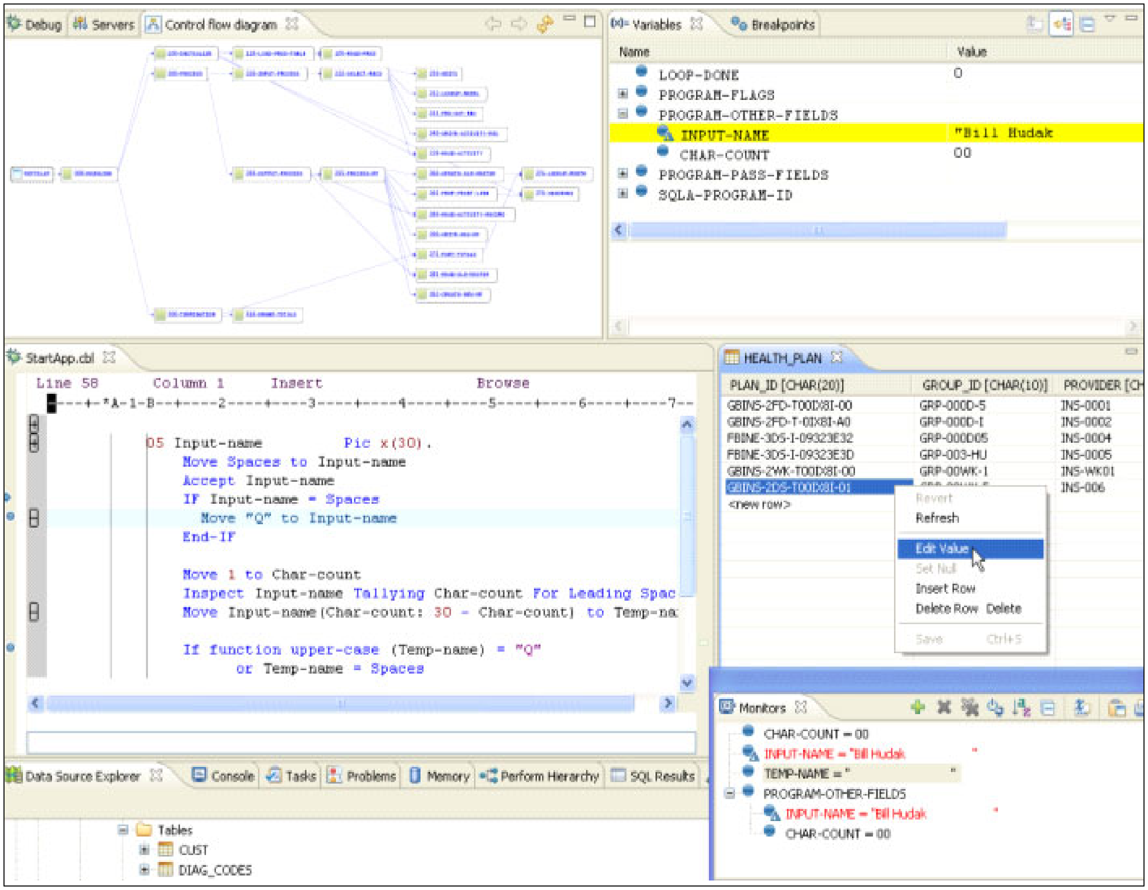


Figure 3-7 Debug perspective with customized views enabled

3.5 Eclipse and ISPF comparison

At first glance, you might think that ISPF and Eclipse tools, such as Rational Developer, IBM Rational Team Concert for System z, and IBM Rational Asset Analyzer, are totally dissimilar, but that is not true. They have these similarities:

- ISPF has menus, such as the Primary Option menu and the menus at the top of each panel (Figure 3-8 on page 78).
- Rational Developer has menus, such as the context menus and the menus at the top of the workbench.
- Rational Developer has views that are based on Eclipse views.

- ISPF also has views, although they are called *options*, *panels*, or *screens* (Figure 3-8).
- Rational Developer has perspectives.

ISPF is the development tool. Think of SQL Processor Using File Input (SPUFI) as the data perspective and the other custom tools that are installed in your shop as additional perspectives, which consist of one to many views (or options).

So, even though the names of the software development components differ, the functionality that is provided is equivalent for ISPF or Eclipse (Figure 3-9).

The learning and mastery curve is mapping what you know and do in ISPF to new tools that are found in other places (separate views and perspectives). We make every attempt in this book to tie the new information back to z/OS and ISPF wherever analogies exist.

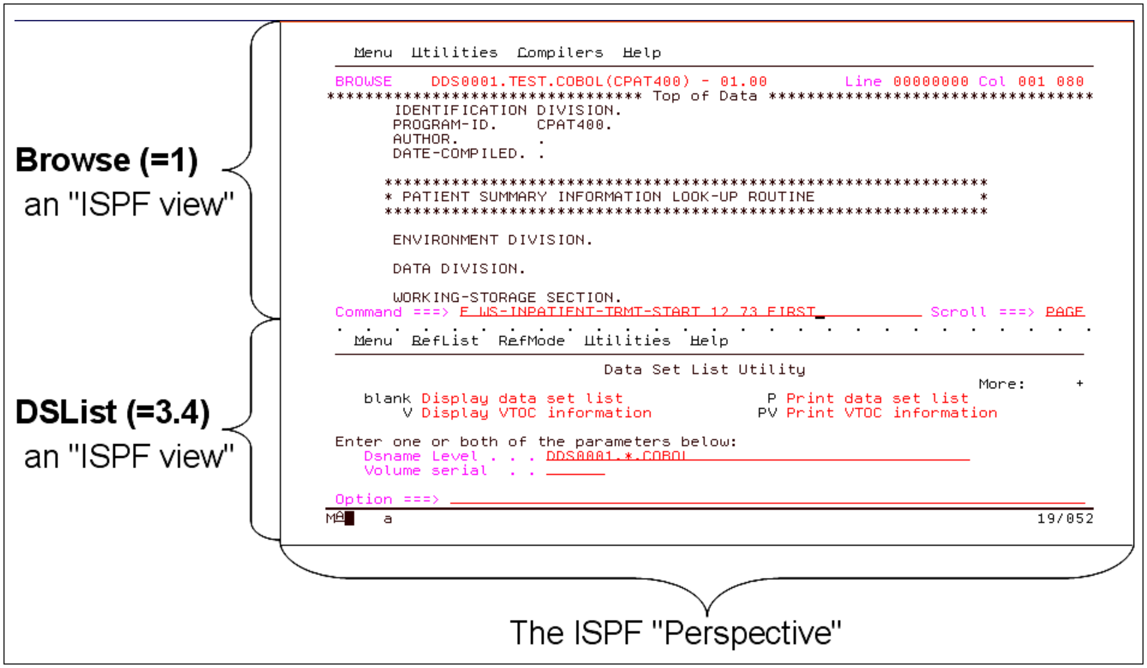


Figure 3-8 ISPF in split screen where each split-screen option is the equivalent of a "view" and the ISPF session is a "perspective"

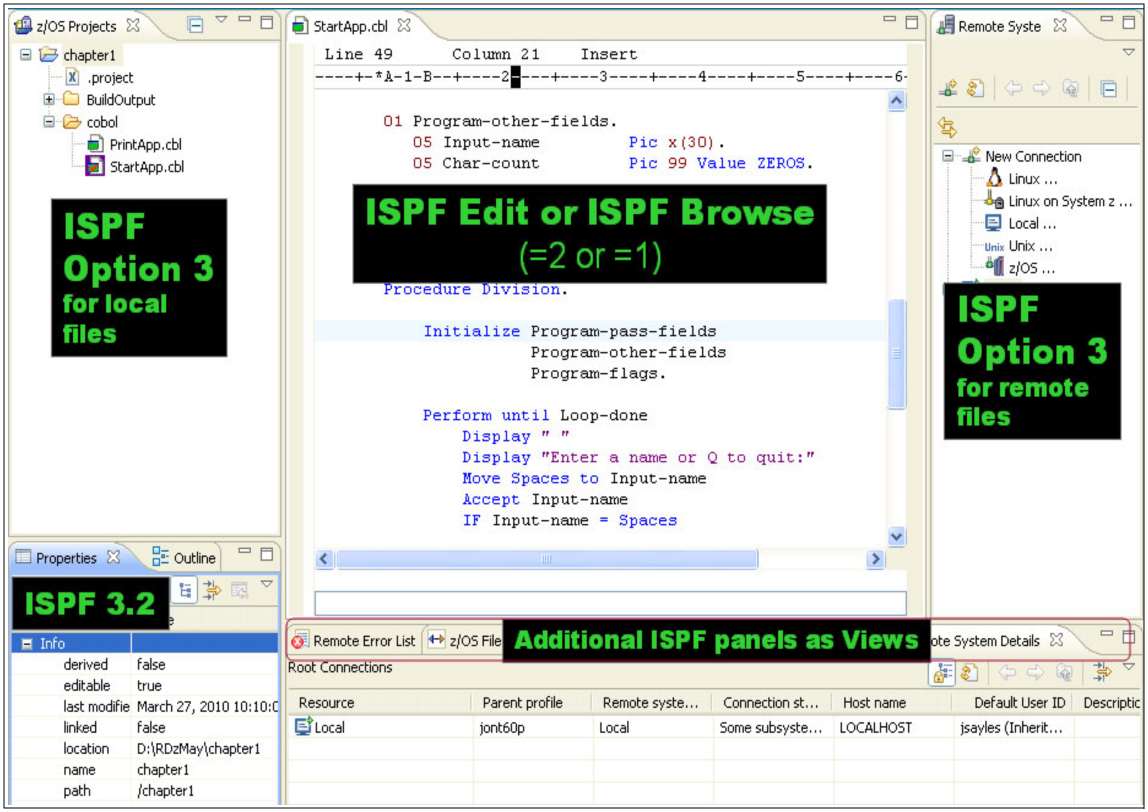


Figure 3-9 Several ISPF panel equivalents of Eclipse

3.6 Eclipse and property files: Customizing your development options

In this section we discuss various ways to customize the way you work.

3.6.1 Options for customization

In ISPF, you have several ways to customize the work that you perform within your development environment:

- You can set up or customize your PF keys from ISPF option 0.
- Your systems programming staff and database administrators (DBAs) can customize the job control language (JCL) that you use to compile, link, build, and run your applications.

- Your systems programmers can customize your ISPF profile to include various IBM, custom in-house, and third-party Time Sharing Option (TSO)-based modules in your ISPF profile data set.
- You can also customize the editing options for each program dynamically (by typing PROFILE from the command line).

In Eclipse products, you can also customize the way that you work. Eclipse provides two product components for this customization:

- Preferences option, which is similar to your developer-level customization of your PF keys and editing options
- Property groups option, which is similar to the compiler and build options that are set by your systems programmers

Both of these customization options are straightforward, as long as you understand their functions and what the various options mean, just as when using ISPF, you need to understand how the COBOL NUM ON setting affects your source code *before* you use it.

We describe various preferences and property groups in the context of the tools in the IBM Enterprise Modernization (EM) Workbench as you learn about them in subsequent sections.

3.6.2 Why learn to use Eclipse-based tools

If you are an experienced ISPF developer who is competent and productive using green-screen technology, you might wonder about adopting Eclipse-based technology. What is in it for you? Eclipse actually offers you many benefits at any stage in your career:

- You enhance your resume with leading-edge technology.
- You make many aspects of your current work easier and simpler, because you use a windowed UI, which is a better use of the available space, allowing you to see, edit, and work with large areas of your programs concurrently. For example, you can view 190 lines of source at-a-glance.

And, the IBM workbench offers advanced application tools, which provide the following benefits:

- Significantly minimize the amount of typing that you have to do in all phases of your development work, from scoping and analysis through unit and integration test
- Simplify navigation and scrolling

- Provide advanced test data manipulation facilities, simplifying the amount of work to build out your test suites
- Replace program coding and development with responding to wizards and prompts

Eclipse is a dynamic new environment, much like ISPF was in the late 1970s when it was first introduced. Many software vendors are extending their functionality to Eclipse, and you can benefit from their efforts to bring increased function and value to Eclipse.

Eclipse functionality can help you shorten the time that you spend on daily tasks, which allows you to focus on programming, maintenance, and analysis.

3.7 Summary

In this chapter, we have presented Eclipse from an ISPF developer's point of view. We have described the following fundamental Eclipse terms and concepts in relation to ISPF functionality:

- Eclipse
- View
- Perspective
- Menu
- Editor

Eclipse technology can make your job easier and more fun.

4

IBM Rational Developer for System z

This chapter describes IBM Rational Developer for System z and its features, functionality, terms, and vocabulary, from the viewpoint of a z/OS Interactive System Productivity Facility (ISPF) developer. If you already use Rational Developer at your company or if you used it in college, you might not need this information and be able to skip this chapter. But, if you have never used Rational Developer, read this chapter to get the most out of Part 3, “The tools in the optimized life cycle: Practitioner” on page 65 and Part 4, “z/OS application maintenance and support” on page 473.

This chapter is specific to z/OS maintenance and support activities, such as System z development and maintenance tools and the integrated development environment (IDE) of the IBM Enterprise Modernization (EM) Workbench. We do *not* describe the many other significant functional areas of Rational Developer, so that we can focus on z/OS traditional maintenance. Also, we do *not* describe the following Rational Developer key value technologies (for more information about these technologies, see “Related publications” on page 601):

- Web Service generation through a wizard-driven process
- DB2 stored procedure generation through a wizard-driven process
- Data flow modeler
- Unified Modeling Language (UML) to COBOL generation through a series of wizards and steps

4.1 Rational Developer

Rational Developer is the IDE of IBM EM Workbench, uniting z/OS resources with desktop software for developing, maintaining, supporting, and modernizing your z/OS-based applications. Rational Developer is built on the Eclipse framework. As shown in Figure 4-1, you can see that Rational Developer is a comprehensive package with a rich set of modern IDE-based development capabilities:

- Access to z/OS assets: Organizing, allocating, managing, editing, and working with z/OS data sets and tools.
- Development, maintenance, and support of business application functionality (analyzing, editing, and compiling or assembling) and source files using integrated language-sensitive editors for COBOL, PL/I, IBM High Level Assembler (HLASM), C/C++ and Java, job control language (JCL), basic mapping support (BMS), and Message Format Service (MFS).

  The editing process installation is direct to z/OS partitioned data sets (PDS libraries) by default, or you can configure the setting to access source through most of the source control management (SCM) systems on the market.
- Managing and editing unit test data, such as queued sequential access method (QSAM) or sequential data sets, Virtual Storage Access Method (VSAM) or indexed, relative, and entry sequenced data sets, DB2 tables and views, and Information Management System (IMS) databases.

  Note that access to DB2 and IMS data is through Java Database Connectivity (JDBC), which lowers the cost of performing routine test data development and maintenance considerably.
- Access to z/OS Job Entry Subsystem (JES) submitting jobs and JES spool files, organizing and managing batch jobs. Rational Developer can also provide access to your in-house and third-party installed z/OS software.
- Wizards and *declarative development* or software tools that replace low-level coding to create, deploy, and test DB2 stored procedures, Web Services, and Service Component Architecture (SCA), transforming Unified Modeling Language (UML) models to COBOL, and using Enterprise Generation Language (EGL) development of Web 2.0 and Java 2 Platform, Enterprise Edition (J2EE) applications.
- Integration with the tools in the rest of the EM Workbench, such as Rational Asset Analyzer for System z and Rational Team Concert for System z.

Rational Developer IDE-based development features, when used for maintaining and supporting z/OS traditional applications, make Rational Developer a flexible yet robust product that supplies veteran and entry-level ISPF developers with tools that can perform the same functions as ISPF, except better.

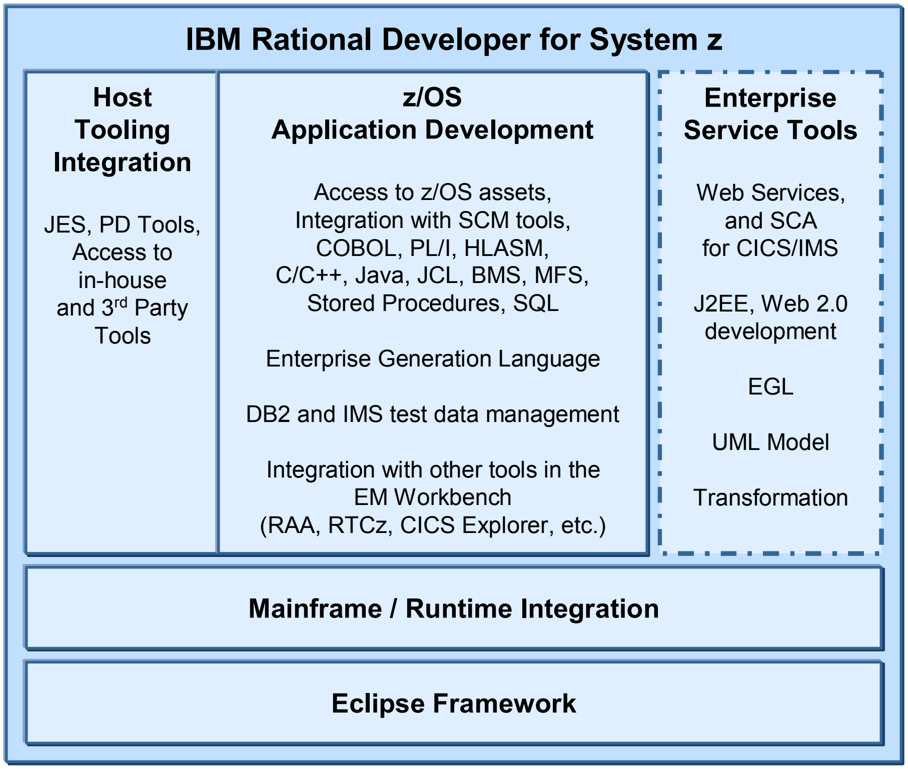


Figure 4-1 Overview of Rational Developer software architecture

In Figure 4-1, you can see that Rational Developer is rich in functionality. In fact, attempting to document all the features of Rational Developer is beyond the scope of this book. We focus on the tools on the host tooling integration and z/OS application development sections in Figure 4-1.

In a subsequent section, we briefly describe a few of the Enterprise Modernization tools, especially in the area of Web Services development, because Web Services development is such a common application requirement. However, we consider topics, such as Service Component Architecture, Enterprise Generation Language (EGL), which is used to develop J2EE and Web 2.0 applications, and UML model transformation, beyond the scope of this book.

4.1.1 Rational Developer terms and concepts

If you are an experienced z/OS developer and if you have read the previous section about Eclipse, there is not much more that you need to know to understand Rational Developer.

Figure 4-2 shows several common Rational Developer views that are used during a typical COBOL analysis or edit session (these numbers relate to the numbers in Figure 4-2).

(1) The program BNCHS601 was opened in the COBOL editor from the (2) DDS0001.TEST.COBOL library (PDS), which is shown in the Remote Systems view. (3) The developer uses the Perform Hierarchy view and (4) the Outline view to study the program's control flow (perform chain processing). (5) The developer moves the mouse pointer over and analyzes the declaration of VALID-RECORD, which is a COBOL 88-level data item as shown and is part of the 01-structure FLAGS-AND-SWITCHES.

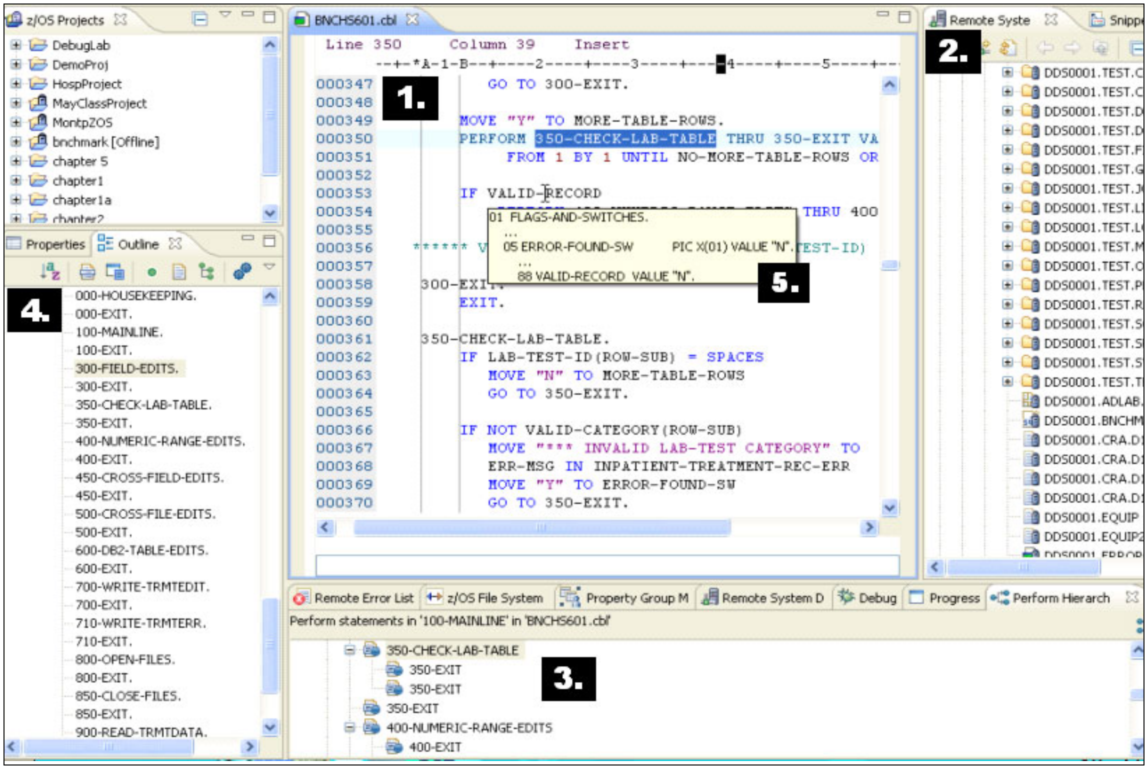


Figure 4-2 Views and LPEX editor functionality of a COBOL program loaded into Rational Developer

Note that the FLAGS-AND-SWITCHES variable can be in the program's WORKING-STORAGE, or it can be in a separate copybook or include source file.

As an Eclipse-based product, the Rational Developer development tools and features include source editors, menus, and views and perspectives. You can customize all these editors, views, perspectives, and menu tools extensively. We discuss customization later.

4.1.2 Source editors

In the Eclipse section, we described several characteristics of the Rational Developer source editors. Rational Developer launches the appropriate editor for the source file that you want to open based on the language extension of the file or z/OS data set that is mapped to a specific extension. For COBOL, PL/I, HLASM, C/C++, Java, JCL, BMS, MFS, and database description (DBD)/program specification block (PSB) source, Rational Developer editors provide the following features. However, the features and functions vary by language and option:

- In-depth language-sensitive development:

  Figure 4-2 on page 86 shows the COBOL colorization. Additionally, in-depth language features exist for Intellisense typing using hot keys to develop by selection from variable and paragraph name lists, syntax validation on several levels, and language semantics. In Figure 4-2 on page 86, you can see the Perform Hierarchy view, which shows the PERFORM chain paragraph flow as indented paragraph names in the view. All the supported languages have operational language sensitivity.

- Hyperlink integration:

  With COBOL local and remote compiler output, with the EM Workbench analysis and team development tools (see Figure 4-3 on page 89 showing the integration between Rational Developer and Rational Asset Analyzer using Rational Asset Analyzer integration), and with most major source control management systems, you can build seamless processes from your current development workflow to Rational Developer. All the supported languages have operational integration, although with various tools and functionality

- Multiple editor identities:
  - Separate editing profiles exist that you can use in your work, such as a pure COBOL editor, Live Parsing Extensible (LPEX) editor, ISPF, XEDIT, vi, and so forth.
  - Figure 4-2 on page 86 shows an ISPF profile with the prefix area on the left and the ISPF command line immediately under the source. Rational Developer provides this ISPF profile for convenient skills transfer for ISPF mainframe developers.
  - In addition to the ISPF profile, eight other editor profiles exist in the Rational Developer Preferences page (as of writing this book). You can change the editor profile at any time and to any supported language.

- Feature-rich:
  - If you are an ISPF mainframe developer, you can perform the majority of your regular z/OS tasks by using Rational Developer. And, you can do much more with Rational Developer than with ISPF. We describe the Rational Developer functions later in this section. However, for an initial look at Rational Developer, see Figure 4-2 on page 86, and note the Hover Help feature that is used to understand a variable's declaration quickly.
  - Consider the potential benefits of analyzing a program that you need to understand to maintain or support its functionality (Figure 4-3 on page 89). Rational Developer supports various languages with separate analysis and editing features, depending on their purpose. For example, the control flow diagram for a COBOL program is replaced with a visual batch diagram for JCL (see Chapter 5, "Rational Asset Analyzer" on page 199).
  - As of this writing, Rational Developer does not support all the languages with the Rational Asset Analyzer integration functionality that is shown in Figure 4-3 on page 89. Contact your IBM technical support team for a current list.
- Customizable and extensible:

  Using the Preferences page, you can modify most of the aspects of the editors. Using a Rational Developer feature called Menu Manager, you can extend the default menus to include access to many of your in-house or third-party editing tools and capabilities. You can customize and extend all language and editor combinations.

Figure 4-3 shows the integration of the following analysis functions with the Rational Developer source editor through separate Eclipse views (these numbers correspond to the numbers in Figure 4-3):

1. Program diagram view: Shows all called subroutines, DB2 tables, IMS databases, QSAM file components, and VSAM file components
2. Control flow diagram: Shows the paragraph and section execution sequence in a hierarchy chart and integrates with the Rational Developer source editor
3. Rational Developer source editor: Permits you to click a paragraph in the control flow, and the source in the editor synchronizes with this action, allowing you to get a high-level analytical view of the program while learning the details of each paragraph and section
4. A sorted cross-reference view of your variables: Exposes various types of operational activity on each variable and allows you to hyperlink to the variable or to the references to the variable throughout the program.

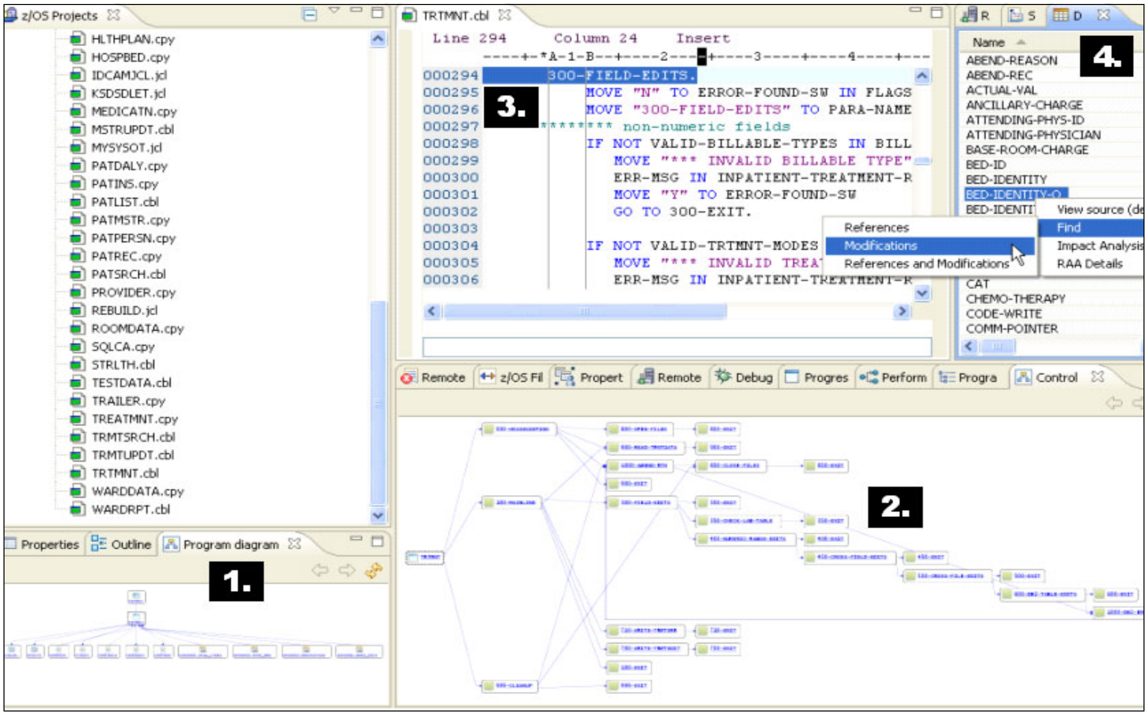


Figure 4-3 Integrating Rational Developer and Rational Asset Analyzer (Rational Asset Analyzer integration)

For example, Figure 4-3 shows that you can use this variable cross-reference to find all statements in the program where the COBOL operation modifies the content of the variable during execution. The list that is produced is hyperlinked with the Rational Developer editor and can be invaluable in solving production ABENDs that occur due to data integrity issues, such as S0C4, S0C7, and so forth.

4.1.3 Rational Developer menus

There are two types of Rational Developer menus:

- Use navigation menus to transfer to areas within Rational Developer to perform a task.
- Use operation menus to invoke a product feature or function.

Tool bar menus and icons

The Rational Developer tool bar menus are based on the industry-standard Common User Access (CUA) design pattern. The rich windowed layout of the Eclipse framework provides a concurrent/parallel development style versus ISPF's serial/sequential development style. The Rational Developer menus and icons (Figure 4-4) provide access to in-place functionality rather than (as with ISPF) the menus acting primarily as navigation options to get to a certain window or sub-window to perform a task.

Figure 4-4 Rational Developer menu and tool bar icons

As you become familiar with Rational Developer's functionality, you can use either the menu options (File, Edit, Search, and so forth) or the tool bar icons, which are listed directly beneath the menu and have associated hot key definitions. The tool bar icons provide access to the following tasks:

- Saving and printing files
- Accessing typical development functional sub-menus:
  - Importing/exporting projects and files
  - Search
  - Debug
  - Various Eclipse perspectives:
    - Debug perspective
    - Data perspective for working with DB2 and IMS test data
  - The Help system
- Navigating to areas with which you are working in the source:

- By clicking the left facing arrow and right facing arrow, as shown in Example 4-5, you can move in the following ways:
  - Return to the last line on which your cursor set focuses, which is similar to the navigation concept of breadcrumbs on a web site
  - Return to the last line that you edited where you modified your source
  - Go forward and backward or to or between points of interest in your program

The tool bar icons provide only a portion of the Rational Developer menu functionality. There are no tool bar icons for the following functions, but you can use the menu options for these functions:

- Customizing preferences
- Opening views
- Opening and resetting to the installation defaults (perspectives)
- Searching for text within the current source file

The tool bar enables you to use easy single-click tool bar options to access the most commonly used functions.

Context menus

Rational Developer supports the notion of *context*, which means whatever has been selected.

Rational Developer context menus provide access to available functionality that allows you to act on whatever source line or group of source lines you have selected. Figure 4-5 shows the action of selecting multiple source lines (in this case, an SQL single-row SELECT statement) and then using the context menu in the editor to comment out all selected lines in one operation.

Without increasing the volume of menu options and tool bar icons and without forcing you to type commands, context menus provide useful or *intelligent* access to development functionality in one or two mouse-clicks, always starting with the right mouse button, to launch the context menu. Context menus are also a graphical IDE standard with most software tools on the market.

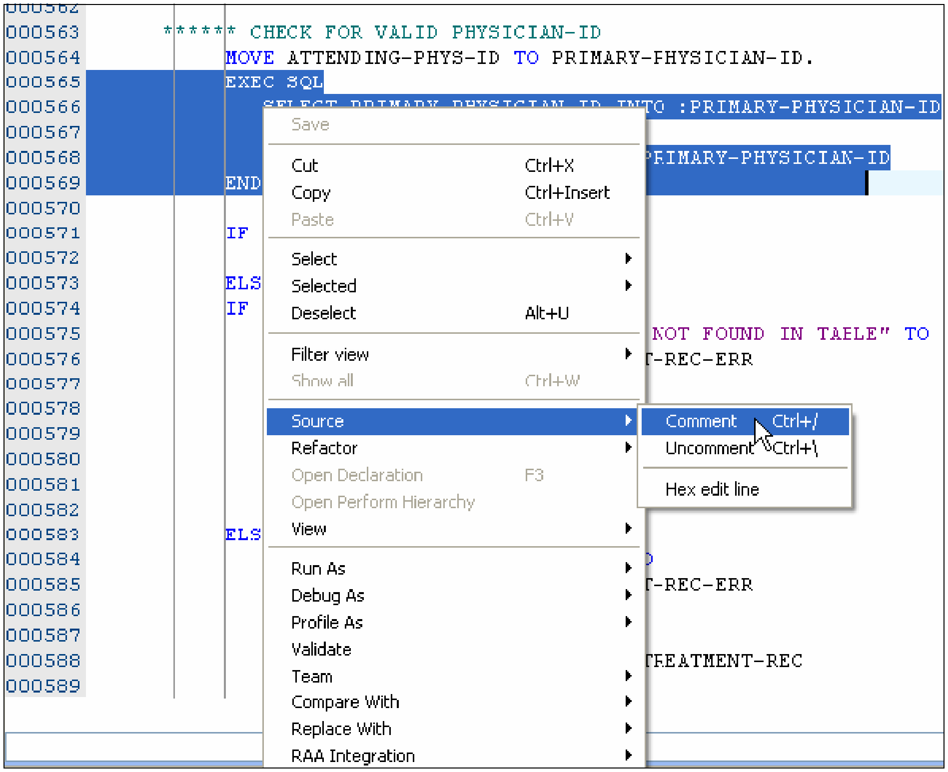


Figure 4-5 Rational Developer context menu option to comment all selected lines

4.1.4 Rational Developer views and perspectives

In Figure 4-2 on page 86 and Figure 4-3 on page 89, you can see that the Rational Developer Workbench presents functionality in Eclipse views that are organized within an Eclipse perspective. A *view* is a small window that enables a single high-level function.

Views

Of the almost 100 Eclipse views, Rational Developer developers performing z/OS maintenance typically work with the following views:

- The Remote Systems view provides functionality for working directly with z/OS resources. Think of the Remote Systems view as equivalent to ISPF option 3 (utilities to work with data sets).

  The following views are under the Remote Systems view:

  – Editor view: Equals ISPF option 2.

- z/OS Projects view: Allows you to organize files that are either off-loaded from z/OS to your local workstation or that were created on your workstation to be uploaded (dragged and dropped) to your z/OS resources later. This view is similar to the Remote Systems view, but instead, it manages and organizes local resources; therefore, many of the z/OS features, such as Allocate, are inapplicable.
- Properties view: Shows filespec (attributes) and z/OS data set statistics. It is similar to ISPF option 3.2.
- Outline view: Provides a high-level look at and hyperlinked access to the divisions, sections, and paragraphs of a program (or the PL/I and Assembler equivalents).
- Perform Hierarchy view: Shows the sections and paragraphs within the COBOL PROCEDURE DIVISION as indented (performed) logic. For PL/I, this functionality is in the Logic Outline Filter. For HLASM, this functionality is in the Branch Instructions Filter.
- Remote Error List view: Holds the syntax error messages' output from a local system's or remote system's compile and hyperlinks to the source line with the syntax problem.
- Property Group Manager view: Provides access to setting and customizing your pre-compile, compile/link/assemble, and runtime options, as well as other properties that standardize your z/OS development.
- z/OS File System Mapping view: Allows you to add or override the defaults for the Rational Developer's assignment of z/OS data set names. For example, if your company's COBOL test source library is named MYCORP.TEST.SOURCE, you assign this file as a .cbl file type. All the language-sensitive functionality within Rational Developer (the editor, syntax checker, upload/download, and so forth) uses this mapping:
  - Mapping a data set name is typically a one-time process.
  - You can map at the data set level. For a partitioned data set (PDS), you can map at the member name level, within your data set.

Perspectives

Views are combined into a *perspective*. Perspectives allow developers to work with a set of views related to a certain high-level task.

In Rational Developer, you commonly use four perspectives:

- z/OS Projects perspective: Combines views that are relevant to z/OS COBOL, PL/I, and HLASM project work
- Debug perspective: Permits you to debug z/OS remote and local applications

- Data perspective: Allows you to perform DB2, SQL, and IMS database-specific work, analyzing database models, testing statements, and modifying DB2 table and IMS segment values
- Enterprise Service Tools perspective: Aggregates views that are specific to building and testing Web Services

To switch from one perspective to another perspective, use the Window menu (Figure 4-6). If the perspective that you are looking for is not an option on the Open Perspective menu, select Other to display a list of all the Rational Developer perspectives.

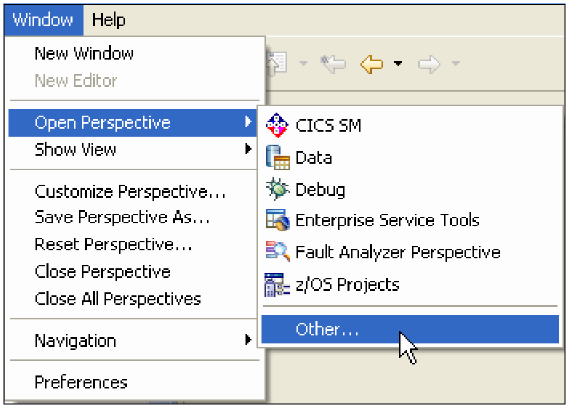


Figure 4-6 Window menu for working with perspectives and views

As shown in Figure 4-6, you can open various Rational Developer views (Show View) or reset the current perspective to the perspective's installation defaults (Reset Perspective). Resetting a perspective is useful if you have closed views inadvertently or stretched the view sizes, and you want to return to the original installation default workbench view.

4.2 Installing Rational Developer

Figure 4-7 shows that you can install Rational Developer into two software platforms:

- z/OS, which is the mainframe to which you connect. You install into a z/OS logical partition (LPAR) to service a number of users that connect to that LPAR to work with Rational Developer client.
- Windows, which is your desktop computer. You install the Rational Developer client software on each developer's desktop computer.

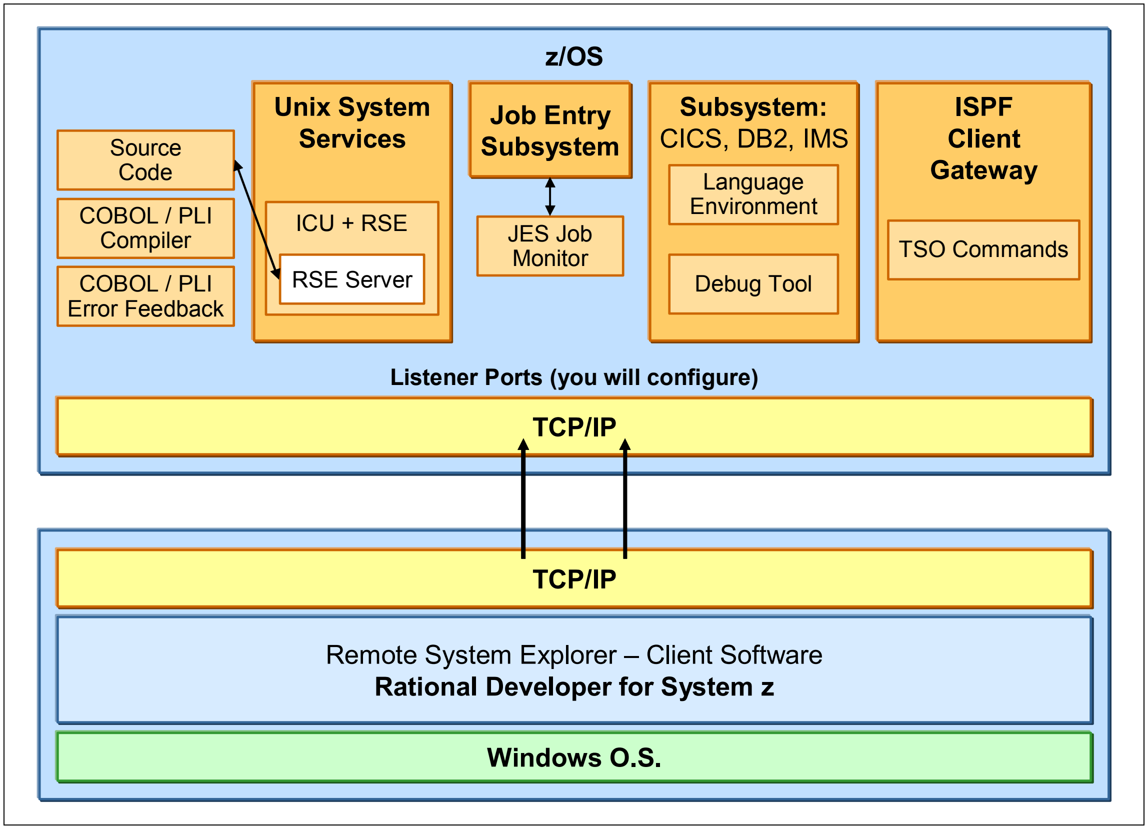


Figure 4-7 Rational Developer component architecture

4.2.1 z/OS installation overview

The z/OS components consist of listeners and interfaces to various z/OS subsystems through UNIX System Services and the Job Entry Subsystem (JES). Your systems programming staff typically installs the z/OS components. The z/OS components require the authority to run jobs that install, configure, and manage entries to the following components:

- PARMLIB and other operating system-level components
- Resource Access Control Facility (RACF), Computer Associates Access Control Facility 2 (CA ACF2), and Computer Associates Top Secret z/OS components
- TCP/IP components

The z/OS components integrate with the IBM Problem Determination Tools Family, CICS Explorer, and any other mainframe integration points.

A dedicated IBM website contains mainframe installation documents and a Quick Start guide to help you through the installation process. As of this writing (and Rational Developer Version 7.6), you can obtain the installation documentation at the Rational Developer Library website:

http://www-01.ibm.com/software/awdtools/rdz/library/index.html

Between Rational Developer versions, the documentation might move to another URL, so we suggest that you use an Internet-based search engine to look for the Rational Developer documentation using the following search text:

- `Rational Developer for System z documentation`
- `Workstation installation overview`

You must install the client software on each Rational Developer developer's personal computer. Rational Developer client runs on the Microsoft Windows operating systems: Windows XP, Windows Vista, Windows Server 2003, Windows Server 2008, and Windows 7. See the latest product installation guide or *IBM Rational Systems Developer, Version 7.0 Installation Guide,* GI11-8355, for an exact list of Rational Developer operating system (OS) levels and required service packs.

Table 4-1 describes your workstation's minimum required hardware configuration. If you use your system for other work in addition to Rational Developer training, consider using the *suggested minimums*. You might need more memory, such as 2 GB RAM, and a higher resolution monitor.

As of Version 7.6.1, IBM supplies the Rational Developer Host Configuration Utility, which is an ISPF panel application that guides you through the fundamental and common optional customization steps for Rational Developer for System z. The application also allows you to execute installation verification procedures (IVPs) and collect debug information.

Table 4-1 Workstation hardware requirements

Hardware	Requirements
Processor	Intel® Pentium® III 800 MHz or compatible processor A stronger processor is recommended.
Memory	1.5 GB RAM Recommended minimum: 2 GB RAM.
Disk space	1.1 GB of disk space is required to install the IBM Rational Developer for System z traditional developer role. 5 GB of disk space is recommended when installing other bundled software. • Disk space requirements can be reduced depending on the features that you install. • Additional disk space is required for the resources that you develop. • Additional disk space is required if you download the electronic image to install the product. • Additional disk space is required if you use FAT32 instead of NTFS.
Display	1024 x 768 resolution using 256 colors A higher resolution and color palette is recommended.
Other hardware	Microsoft mouse or compatible pointing device.

Rational Developer offers many installation methods. Depending on your company’s method of deploying software to its developers, choose from these methods:

- From CDs
- From the web:
  - Using an IBM Passport Advantage® website
  - Using the developerWorks trial download websites
- Using a silent install

Note that the URLs for the web-based installation for these websites can change, so we suggest that you perform an Internet search on `Rational Developer for System z installation`.

The silent install is a custom process. The installation steps are similar for CD or web-based installation. You or the software starts by launching an IBM standard package called Installation Manager. Installation Manager is an IBM Rational standard install package, which is used for many of the popular development products. It provides a simple way for you to update your software, apply “hot” fixes, install software, and uninstall software.

Installing with Installation Manager is a simple wizard-driven process of responding to prompts, selecting installation options, and so forth. You typically use most of the default installation options, but you might consider using these additional options:

- Customize the Shared Package Folder: All IBM-based, Rational-based, and Eclipse-based products that you have on your desktop, as well as many custom plug-in and hot fix installations, use this folder. If the Shared Package Folder is already on your system, the installation typically finds it. If the installation does not find the Shared Package Folder, consider renaming it to a more meaningful name for future reference.
- Check with your company to determine which installation packages to select: The wizard prompts you during the installation to choose any optional features, but these features might be required features for your particular shop.
- Add the Help system as *local content*: You can install the Help system as a series of links to web documents or as local documents. The content is the same content in both places (unless you do not apply the product updates), but with local help, you can access the documentation even if your machine is offline. By default, Rational Developer links to the Internet-based Rational Developer Help content to access the most recently updated content.

4.2.2 Verifying the local installation

When you have successfully installed the Rational Developer client, you must verify that all the local parts work. A simple way to verify that the Rational Developer client works is to create a local project, see that all the examples are in place, edit files, and test other local aspects of Rational Developer client.

Follow these steps to check your Rational Developer client local installation:

1. From the Windows start menu, launch Rational Developer, which might be under an icon for IBM Software Delivery Platform.
2. When the product prompts for a Workspace, enter a meaningful and *empty* folder name, such as c:\rdzworkspace1, or whatever you can write to on your computer.

3. At the Rational Developer Welcome page, close the Welcome tab by clicking the red **X** to the right of Welcome (Figure 4-8).

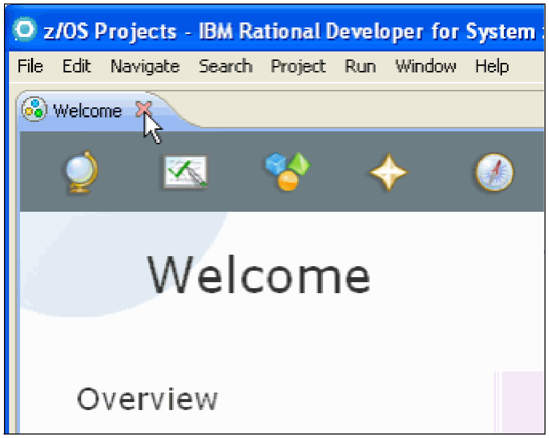


Figure 4-8 Closing the Welcome tab on the initial Rational Developer launch window

4. The product opens to a new, empty workspace (in fact, the workspace you specified a few steps ago when you launched Rational Developer). A workspace is the highest level file folder to which you have access during your development session. It can contain multiple project folders and other resources, such as the sample COBOL project that is shown in Example 4-1.

Example 4-1 Sample COBOL project

```
From the File menu, select:
New    >    Example…
Expand Workstation COBOL Note that if you are a PL/I developer, you
can select a new Workstation PL1 example project instead.
Select:  COBOL Sample 1
Click Next >
Name the project:  chapter1
Be sure a Property Group is selected – this is the default
Click Finish
```

5. To edit a sample program in the project, expand the cobol (or pli) folder that was created by the sample COBOL project example, and double-click a program, which loads the program into the editor (Figure 4-9).

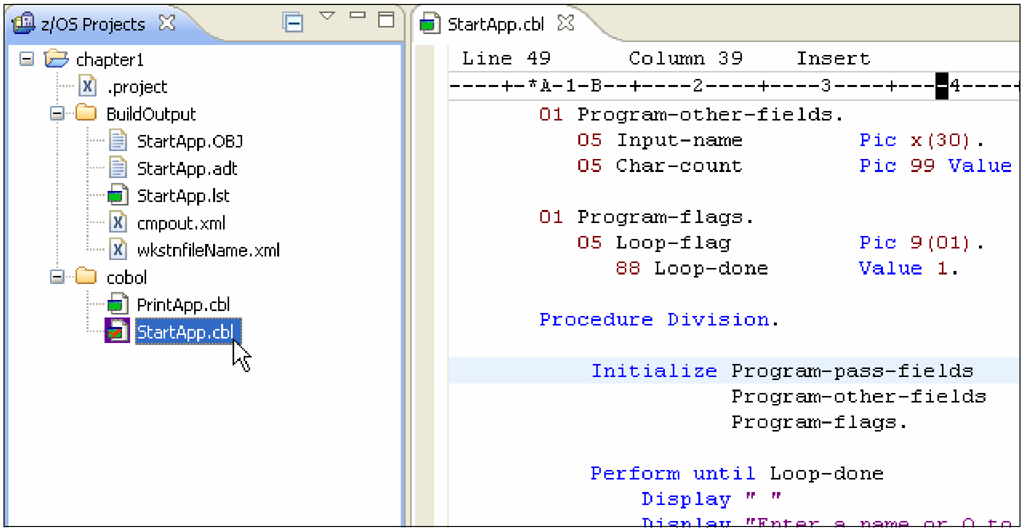


Figure 4-9 Example COBOL project with StartApp.cbl open in the editor

4.2.3 Verifying the z/OS component installation

When your systems programmers indicate that they have successfully installed the mainframe components for Rational Developer, you can test your connection to your z/OS resources, as well as test the z/OS installation. Before you begin, you need the following information:

- The ping-able IP address of your host machine, which is usually obtained from your systems programmer
- The access port numbers that were involved in the Rational Developer mainframe installation, which are obtained from your systems programmers:
  - z/OS UNIX files
  - MVS files and access to the data sets on your Time Sharing Option (TSO) ID
  - Job Entry Subsystem (JES)
- Your TSO ID and password

4.2.4 Connecting to your z/OS LPAR from Rational Developer

Follow these steps to connect to your z/OS LPAR through Rational Developer:

1. Ensure that you are connected to your local area network (LAN) or through you ISP provider through the Internet to a mainframe.
2. From the Remote Systems tab, expand **New Connection** and double-click **z/OS**.
3. From the z/OS UNIX Files wizard form, specify the port number to access z/OS UNIX, and click **Next**.

   For Authentication method, you typically use your RACF/CA ACF2 or CA Top Secret User ID and Password when you connect.
4. From the MVS Files wizard form, specify the port number to access your MVS data sets, and click **Next**.
5. From the JES wizard form, specify the port number to access JES, and click **Finish**, which creates a new connection. It will *not* connect you to your LPAR.
6. To connect to z/OS, right-click over your new connection and select **Connect**. Fill in your User ID (*TSO ID*) and Password (*TSO password*). You need to save your user ID and password. Click **OK**.

   If your z/OS and client installation completed successfully, after a few seconds, you connect to z/OS. You see a set of folders that are similar to but not exactly the same as the folders that are shown in Figure 4-10.

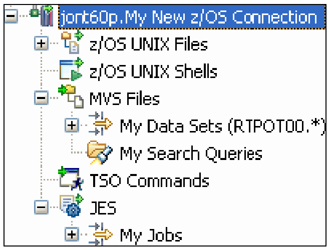


Figure 4-10 Remote Systems Explorer: Initial connection

From the Remote Systems Explorer (Figure 4-10), you can expand the following icons:

- My Data Sets (*Your TSO ID*.*) to see all the z/OS data sets, which have your high-level TSO ID as the first qualifier in the default subsystem name (DSN), that you have allocated.
- JES to see a list of any jobs with your TSO ID as the job owner.

In the next section, we describe the actions that you can take from the Remote Systems view.

4.3 Using Rational Developer

In this section, we show you, the z/OS practitioner, enough about how to use Rational Developer to understand the rest of the materials in this book, at least the content that is Rational Developer-specific.

We do not intend to take the place of legitimate, in-depth product education on Rational Developer, which you can find at this website:

http://www-01.ibm.com/software/rational/education/

And specifically, the Rational Developer course is (as of this writing) at this website:

https://www-304.ibm.com/jct03001c/services/learning/ites.wss/us/en?pageType=course_description&courseCode=RN100

There are additional modes of Rational Developer learning available from IBM and from IBM Business Partners:

- Traditional instructor-led
- Instructor-led online classes that are held over web conferences
- Self-study materials
- White papers, blogs, and articles
- Online resources that are available from the IBM COBOL Cafe:
  http://www-949.ibm.com/software/rational/cafe/community/cobol

Before you begin using Rational Developer to work on your production code base, it is extremely important to get formal and in-depth training. Production coding is so complex, and you are probably already using powerful (if somewhat dated) tools, such as ISPF, Edit Macros, REXX, and so forth. Rational Developer replaces most of these tools and complements the rest of your development tools. Rational Developer requires in-depth training on its features to be able to get productive.

4.3.1 Remote Systems Explorer: Accessing your z/OS assets

Assuming that you have successfully installed both the client and server portions of the Rational Developer software, you can access and manage data sets, submit and manage JCL jobs, issue TSO commands, open a host emulator session, and access REXX commands and many of the third-party ISPF-based tools that you have through the context Menu Manager. The context menu is only a subset of what Remote Systems Explorer offers (Figure 4-11).

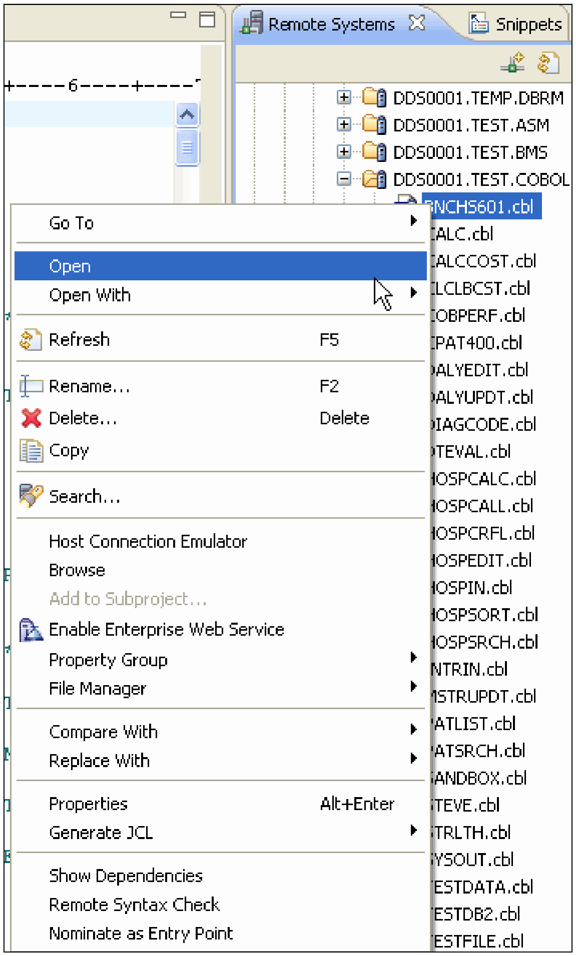


Figure 4-11 Remote Systems Explorer context menu

Figure 4-11 shows what you can do from Remote Systems Explorer when you have selected a partitioned data set (PDS) source member. The development model sequence is the same as with all Eclipse functionality:

1. Select an asset.
2. Right-click, and from the context menu, perform an action with or to the asset that you have selected.

If you are an ISPF developer, you can see that Remote Systems Explorer provides access to data sets (source programs and data files) in Browse or Edit (Open) mode similarly to ISPF primary options 1 and 2. Remote Systems Explorer also provides copy, rename, and delete, which are similar to the common ISPF option 3.*x* data set management functions.

Because Remote Systems Explorer is Eclipse-enabled, you can drag and drop or copy and paste files in these ways:

- Between libraries
- Between TSO accounts on the same LPAR (assuming that you have all the necessary permissions and access rights)
- Between separate LPARs (you can drag and drop files from one physical machine to another physical machine)
- From the mainframe to your local workstation and vice versa

When you open z/OS files using Remote Systems Explorer or drag and drop them, the files are copied from the mainframe down to your workstation. Then, if you copy them elsewhere, the files are copied up from your workstation to their destination, which is another PDS or TSO account on the existing LPAR or another LPAR.

There are limitations on what you can copy and paste using Remote Systems Explorer, for example, you cannot copy and paste a load module (or anything else allocated as RECFM=U).

Access options from Remote Systems Explorer against PDS members include but are not limited to the following tasks:

- Navigate or locate specific members in a large PDS:
  - As you can see from Figure 4-11 on page 103, you can expand a PDS (library) to show all its library members. This feature works well if your PDS has 500 or fewer members. But, this feature does not help if you work with MYDEPT.TEST.SOURCE, or worse, COMPANY.PROD.SOURCE, which might contain up to 100,000 PDS members.
  - Beginning with Rational Developer V7.6.1 (released June 2010), Rational Developer provides the following functions:
    - Customizing the number of PDS members that are listed in the Remote Systems Explorer folder
    - Browsing (scrolling) through the list by using hot key combinations or by using your mouse
    - Locating a specific PDS member
    - Searching for PDS members with names that start with a search text string
- Search: A convenient shortcut to search one or more selected programs and to searching across entire PDS libraries for wildcard text strings; this function is similar to ISPF 3.14 (Search-For) and ISPF 3.15 (Search-For Extended). This function includes searching using regular expressions, which are

powerful search options that are not available from native ISPF. You can also pre-index your components, which makes future searches much faster.

- Compare with: A shortcut to source compare (ISPF 3.12 SuperC). You can compare files across LPARs (the files can exist on separate physical machines). Figure 4-12 shows the source compare UI that is used when *refactoring* programs (changing variable and paragraph names), and at other places in the life cycle.
- Show dependencies: Displays a list of all copybook and include files that are necessary to successfully compile and, by extension, to work with a program.
- Remote syntax check: A shortcut to running a z/OS compile for syntax (this shortcut does not create .OBJ or load modules).
- Replace with local history: Rational Developer manages the source code deltas in your libraries, similar to a source code management (SCM) product, and allows you to recover a file back to a specific prior version.
- Submit if the PDS member you have selected is of type JCL: You can submit from the context menu and follow the job from the Remote Server Explorer/JES functionality.
- Edit or browse and invoking the correct editor on the file type: Rational Developer utilizes a file mapping table, which we describe in the upcoming section, that catalogs the data set types on your mainframe and associates them with one of many Rational Developer editor and workstation file types.

  For example, if your COBOL programs are stored in a departmental library named MYDEPT.TEST.SOURCE, you map this file type to .cbl, and the following actions occur:

  - Remote Systems Explorer shows your PDS members with a COBOL source file icon inside the library.
  - Rational Developer opens the COBOL editor when you browse or open a file within Remote Systems Explorer.
  - Rational Developer opens the File Manager Interface editor when you browse or open a QSAM or VSAM data set.
- Generate JCL: A convenient method of creating development build, or build + batch run JCL, including batch debugging with IBM debug tool. Also, based on the same file mapping table, Rational Developer understands the type of file that is in a PDS and, in conjunction with property groups, performs these tasks:
  - Assembles basic mapping support (BMS) maps and Message Format Service (MFS) files
  - Assembles HLASM file types
  - Compiles COBOL and PL/I programs

Using Remote Systems Explorer with your MVS files offers these functions:

- The Remote Systems Explorer context menu presents options for various of files. For instance, if you right-click over a PDS, you can perform these tasks:
  - Compress the PDS.
  - Index the PDS with a fast and efficient type of search-index (a Lucene index).
  - Add a new PDS member.
  - Allocate a new data set using the PDS file statistics, which are attributes such as LRECL, VOLSER, BLKSIZE, and SPACE allocations.
  - Show all the PDS members in a tabular view, instead of in a list. The tabular view shows additional statistics, such as lines of code, last update date, create date, and so forth. The tabular view of a PDS also allows you to open, sort, browse, compare, delete, and so forth, all the options that are available in the default list view.
- The Remote Systems Explorer context menu approach provides, in one concurrent view, much of the common ISPF data set manipulation functionality with a single mouse click under the consistent context menu usage model. The Remote Systems Explorer context menu makes z/OS access easier to learn for new z/OS developers and makes all z/OS developers more productive.

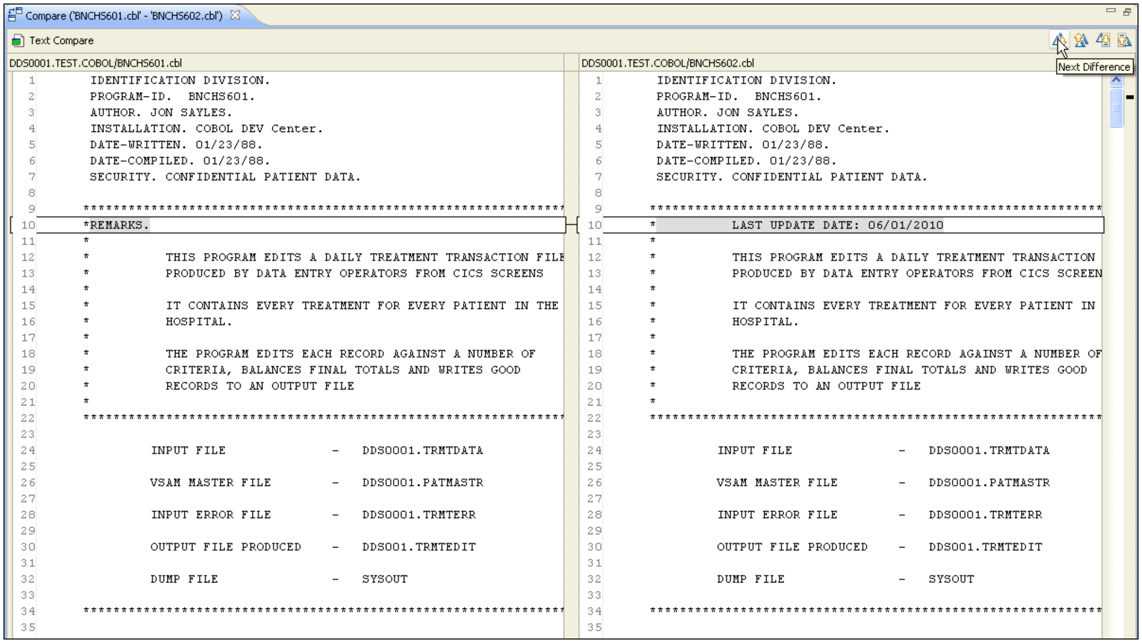


Figure 4-12 Rational Developer source file compare

4.3.2 Remote Systems Explorer and batch job management

Much of the same functionality that is available from Remote Systems Explorer for your z/OS files is available from Remote Systems Explorer for your JES job output. When batch jobs that are owned through their job card to your TSO ID are submitted, you can see their status by expanding the Remote Systems Explorer icon (Figure 4-13).

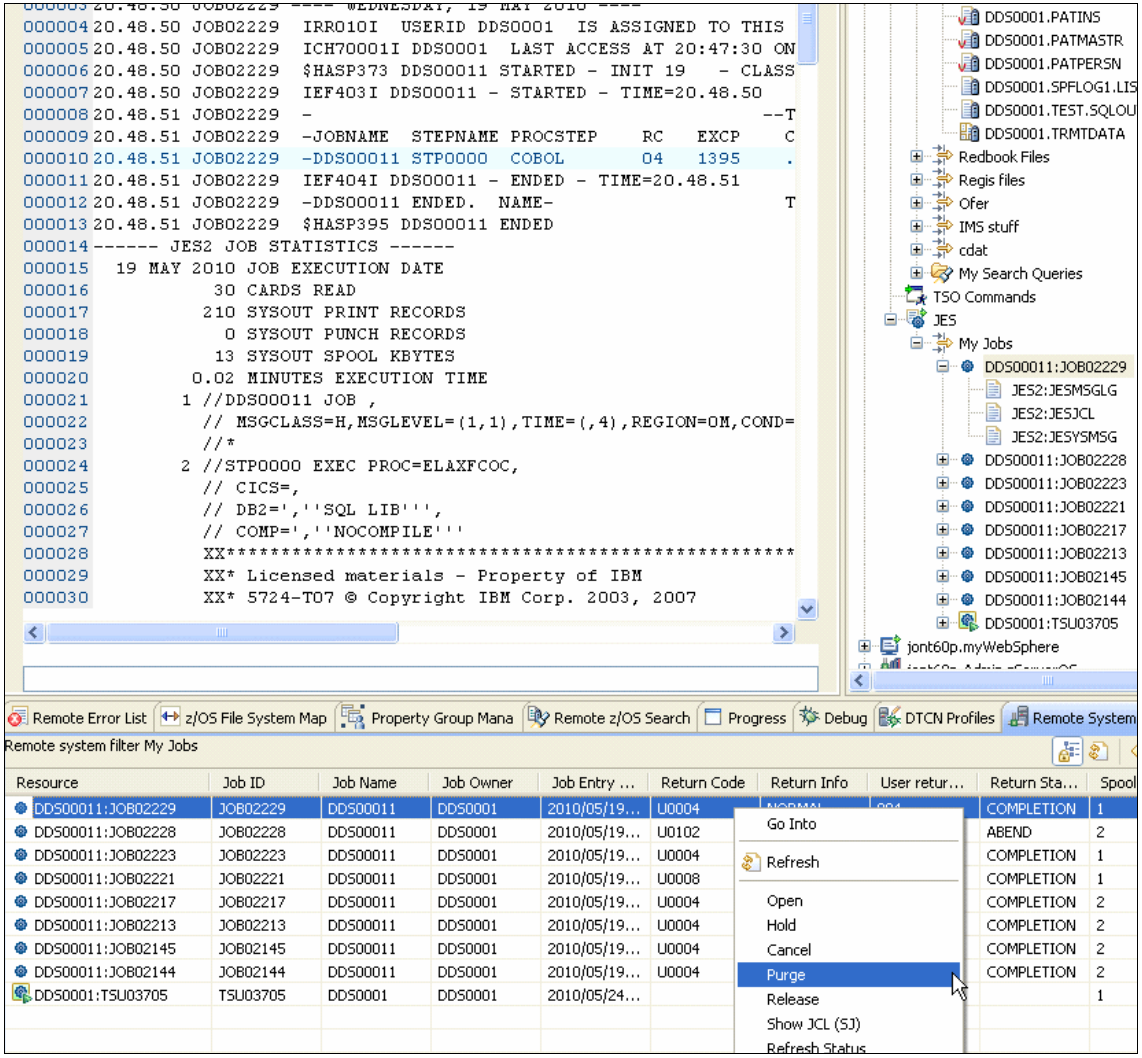


Figure 4-13 Working with batch jobs using the JES feature of Remote Systems Explorer

You can perform the following tasks, as shown in Figure 4-13 on page 107:

- Open jobs to view their output. You also can view the output of individual job steps.
- Cancel submitted jobs that have not yet run (these jobs are identified by a green triangle pointer in the icon, for example, Job ID TSU03705 in Figure 8-14).
- Purge jobs that have completed.
- Release jobs in TYPRUN=HOLD status, which perhaps are waiting for other jobs to complete and to release file resource locks.
- Show JCL, which opens the original JCL file (that you submitted) in the Rational Developer editor.

When you view your jobs in tabular format (Figure 4-13 on page 107), you can sort the jobs based on any of the column headings in the table, as well as perform these operational functions against the jobs.

4.3.3 Remote Systems Explorer and data set filters

By default, Remote Systems Explorer permits you to see and work with files (and batch job JCL listings) that you own because your TSO ID is in the high-level data set name qualifier, or for jobs, in the JOB card. Next, we explain how to view and work with other z/OS assets by using Remote Systems Explorer filters.

Filters are similar to an ISPF DSLIST/REFLIST file search result collection, but filters are made persistent, named, and given a folder icon in the Remote Systems Explorer view. Filters extend what you can do with Rational Developer and simplify certain organizational problems, by providing a flexible means of accessing someone else’s data sets and jobs (Figure 4-14).

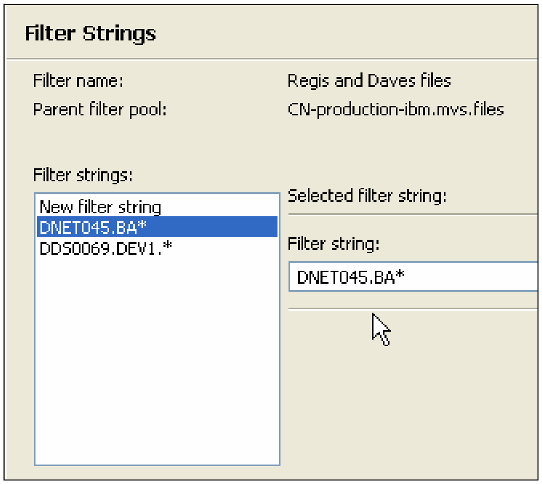


Figure 4-14 Filter strings

As shown in Figure 4-14, you create filters by defining wild card text string patterns that are used to locate the files that you want to filter into your folder. This function is similar to ISPF DSLIST (3.4) if you made the DSLIST results a permanent organized list of files with data set names that conform to the filter strings.

In Rational Developer filters, the data sets that you want to view are displayed in an Expand/Contract Remote Systems Explorer folder entry (Figure 4-15). You can have as many filter strings as necessary to pinpoint the data sets that are organized within the filter (see Figure 4-14, and, specifically, note the New filter string option). Rational Developer uses OR logic with the multiple filter strings, therefore, displaying all data sets that matching any wild card text string in the filter.

By using filters, you can perform these tasks:

- Create data set lists (folders) under a separate high-level qualifier.
- Access another TSO user's data sets, depending on your authorization.
- Create a single expand/contract entry in Remote Systems Explorer that contains all (or most of) the data sets that you need for a task or for your project work.

- Perform these same tasks for your z/OS batch jobs. You can define JES job filters through wild card text strings, which organize your jobs in these categories:
  - Job ID
  - Job owner
  - Job status, that is, my finished jobs
  - Job class
  - Job output class
  - Other JES attributes

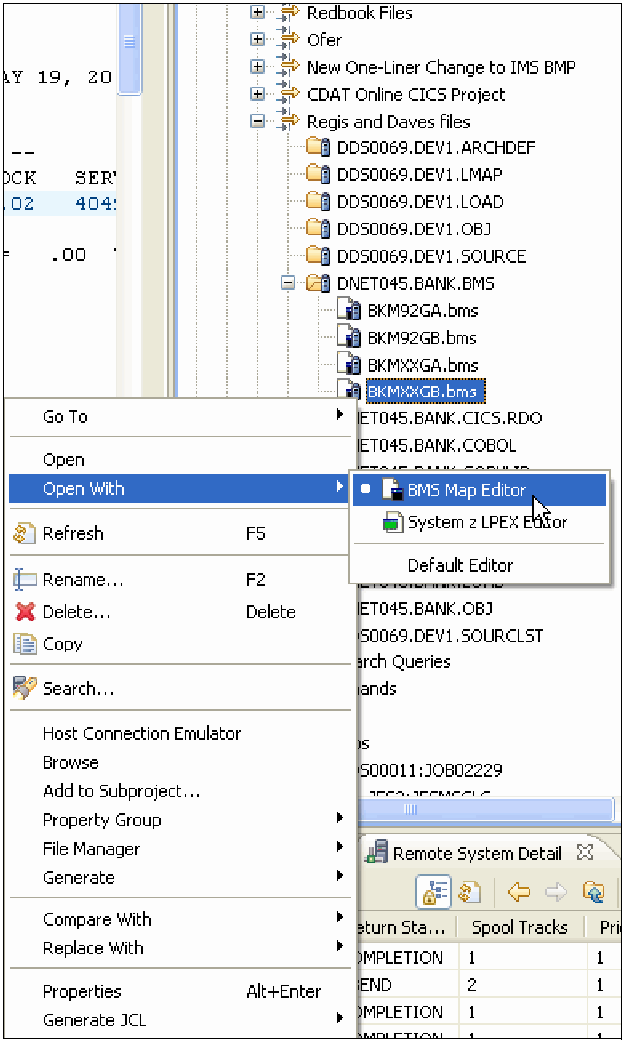


Figure 4-15 Remote Systems Explorer data set filters showing an expanded filter and accessing a file within a filter

4.3.4 Rational Developer source editors

Earlier, we described Rational Developer source file editors:

- Language-sensitive
- Integrated with other tools in the workbench
- Extensible (customizable to your development style and tastes)
- Feature-rich
- Multi-faceted (they support various profiles, such as an ISPF profile, an XEDIT profile, vi, or eMacs

In this section, we provide examples and usage instructions.

Rational Developer uses the Live Parsing EXtensible (LPEX) editor as its installed editor. IBM has customized the LPEX editor significantly to make it simple to use, powerful, and flexible (flexible refers to the EXtensible part of the name).

4.3.5 Language-sensitive editing: The LPEX editor

Language-sensitive editing exists in two areas:

- At the file level, Rational Developer opens the appropriate editor for you, based on the file type:
  - COBOL files open a COBOL editor
  - PL/I files open a PL/I editor
  - JCL files open a JCL editor
  - Basic mapping support (BMS) files open a BMS editor
  - MFS files open an MFS editor

- Inside the Editor view, LPEX applies the live parsing aspect of its name to your source, including but not limited to the characteristics that are shown in Figure 4-16.

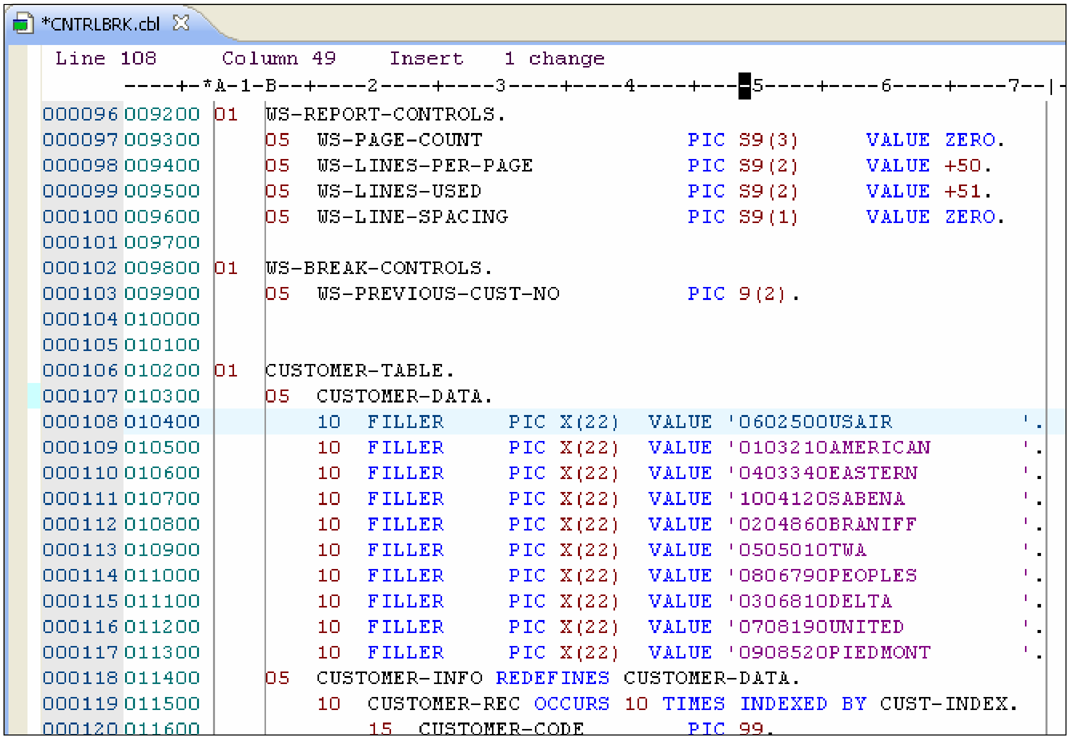


Figure 4-16 COBOL editor

Parsing areas

Figure 4-16 shows the following live parsing areas:

- At the top of the editor is a column ruler, and two vertical coding area lines extend downward (marking the COBOL A and B margins).
- The current line has a light-blue background, and the column ruler shows the current cursor position (reverse video).
- The top line over the ruler also shows the current cursor position, whether the personal computer is in insert (as opposed to overstrike) editing mode, and the number of source changes made during this edit session.
- The ISPF prefix area (for edit commands) is on the far left

- Various parts of the source are shown in specific colors:
  – Keywords are blue.
  – Identifiers, such as variables, paragraph names, and section names are black.
  – Literals, in double quotation marks or single quotation marks, are maroon.
  – Comments are green (not shown).
  – COBOL sequence numbers (columns 1-6) are green.

You can customize all these editing features by selecting **Windows** → **Preferences** → **LPEX Editor**.

Content Assist and hover

Beside the standard layout and colorization of the LPEX language editors (COBOL, PL/I, HLASM, SQL, BMS, MFS, DBD/PSB, and so forth), all the editors provide a common Intellisense typing called *Content Assist*.

By placing your cursor in the source, and pressing Ctrl+spacebar, a list of valid syntactic keywords and identifiers appears (Figure 4-17), allowing you to select, rather than type, the variable, paragraph, or language verb that you want. In addition, you can select fully qualified verb constructs, and Content Assist shows you the variable's definition and source (in COBOL, the 01-field to which the variable belongs) as an additional measure of clarifying the statement.

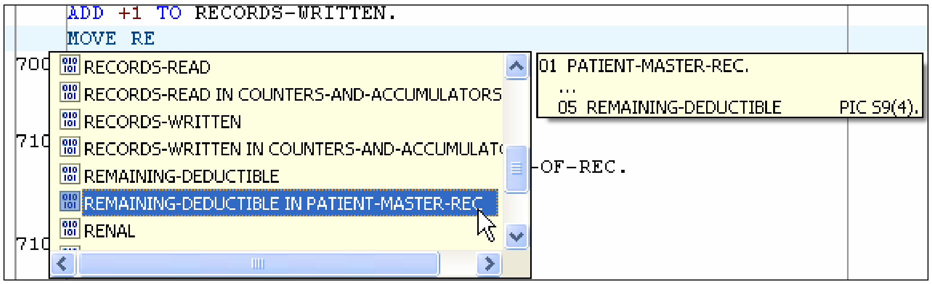


Figure 4-17 Content Assist activated by pressing Ctrl+spacebar

4.3.6 Viewing, editing, and navigating options

Rational Developer provides useful programs to help your understanding, navigation, and viewing. There are specific views, such as the Outline view (Figure 4-18 on page 115), that show your program's divisional areas and a list of variables and paragraph names. The Outline view is like an X-Ref from a compile listing with the added benefit that all Outline view entries are hyperlinked back to their source line, so clicking an Outline view entry for a paragraph name navigates directly to the paragraph and also for variables.

Figure 4-19 on page 116, Figure 4-20 on page 117, Figure 4-21 on page 118, and Figure 4-22 on page 119 show a few of the types of useful split screen view editing features, although not all of them, that are available to provide you with the necessary tools to scale up to the requirements of complex, cross-language, and cross-platform application work:

- Figure 8- 20 shows working with a program (in split screen mode) and several of its copybooks. Each windowed pane in the workbench is a separate physical window-editor instance, providing full edit (even in split screen on the same source file) and full access to all the Rational Developer editing tools.
- Figure 8-21 shows editing a COBOL/DB2/CICS program with full edit to the CICS BMS map editor and the DB2 table editor. All split views are the full tools open on the various source files.
- Figure 8-22 shows editing a z/OS batch job, with the run stream JCL filtered by EXEC statements except for the step that we want. Additionally, we filter (exclude) all references to a VSAM file named PATINS along with viewing a program and VSAM copybook for this same file.
- Figure 8-23 shows debugging a COBOL program, displays the program's control flow, which is available using Rational Asset Analyzer integration with Rational Developer, and shows the program's variable values that it has monitored (a selected subset of the variable values). Even during debug, you can pause and modify a DB2 table, which allows you to test various cause and effect scenarios while remaining under debug. The DB2 table can be a QSAM file, a VSAM file, or an IMS database.

Opening these multiple views shows that by using Eclipse windowing technology, you can perform related analysis and editing tasks concurrently, through simultaneous views of related source code. Without this capability, you are limited to using one tool after another in a sequential manner. With Eclipse windowing technology, you can increase your editing development productivity through concurrent views and optimized screen area.

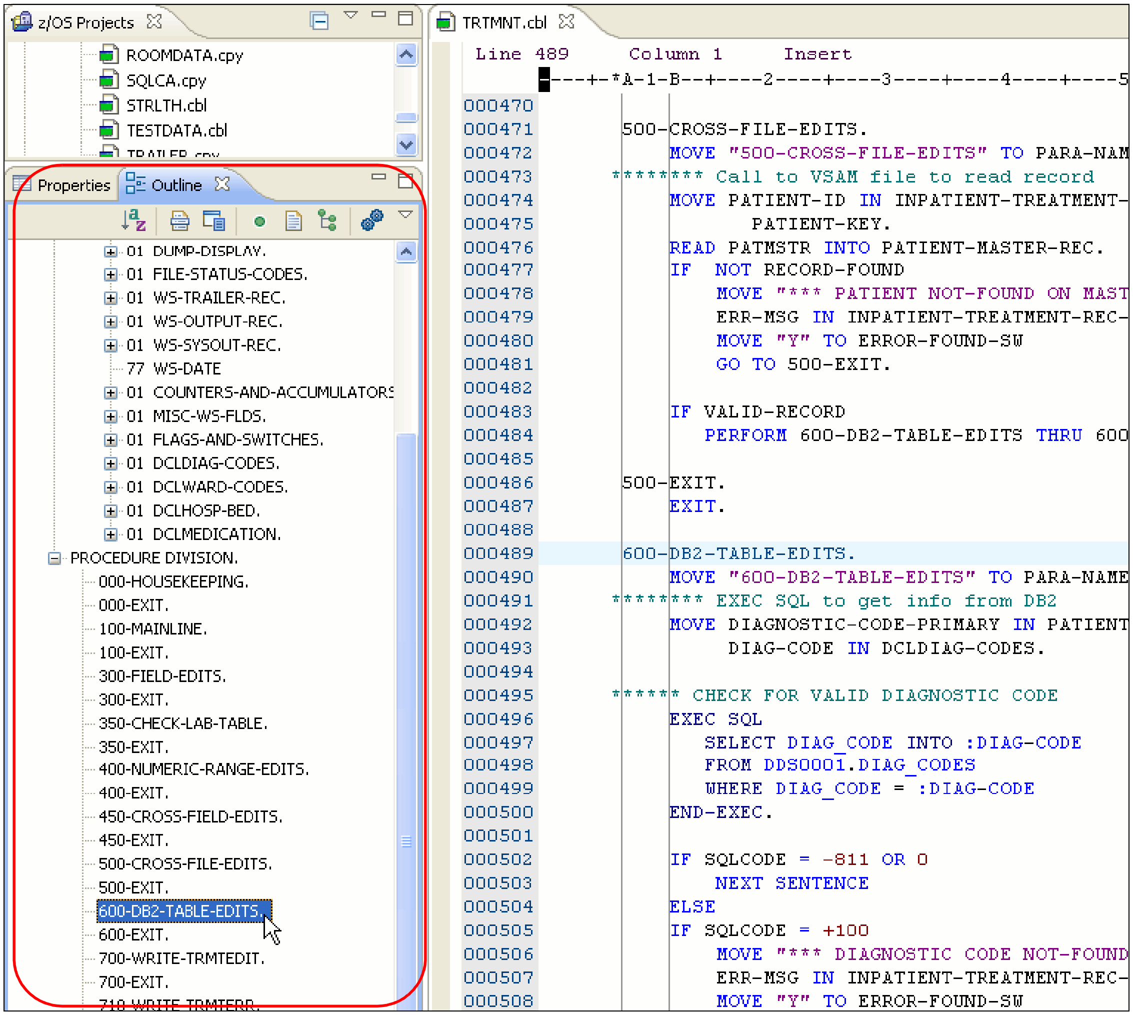


Figure 4-18 Outline view that is used for efficient program navigation

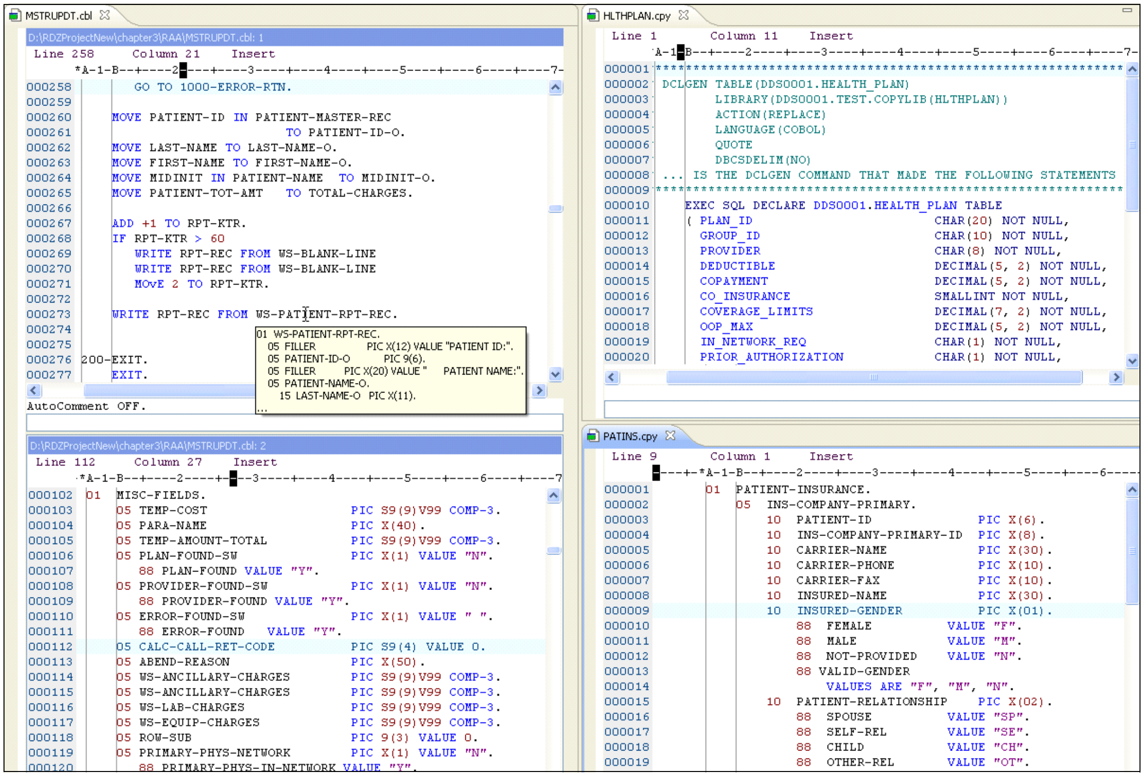


Figure 4-19 Editing a program in split screen with two copybooks edited in separate physical windows

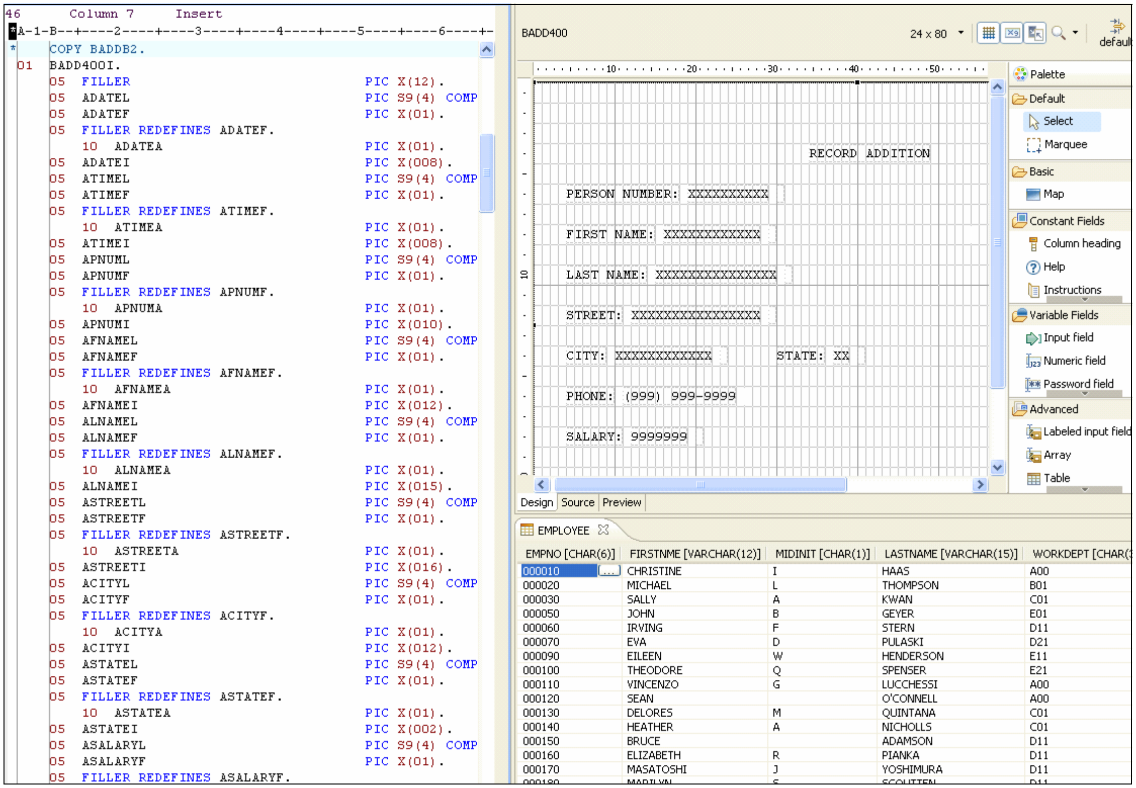


Figure 4-20   Editing a CICS DB2 program while concurrently working with its BMS and a DB2 test table

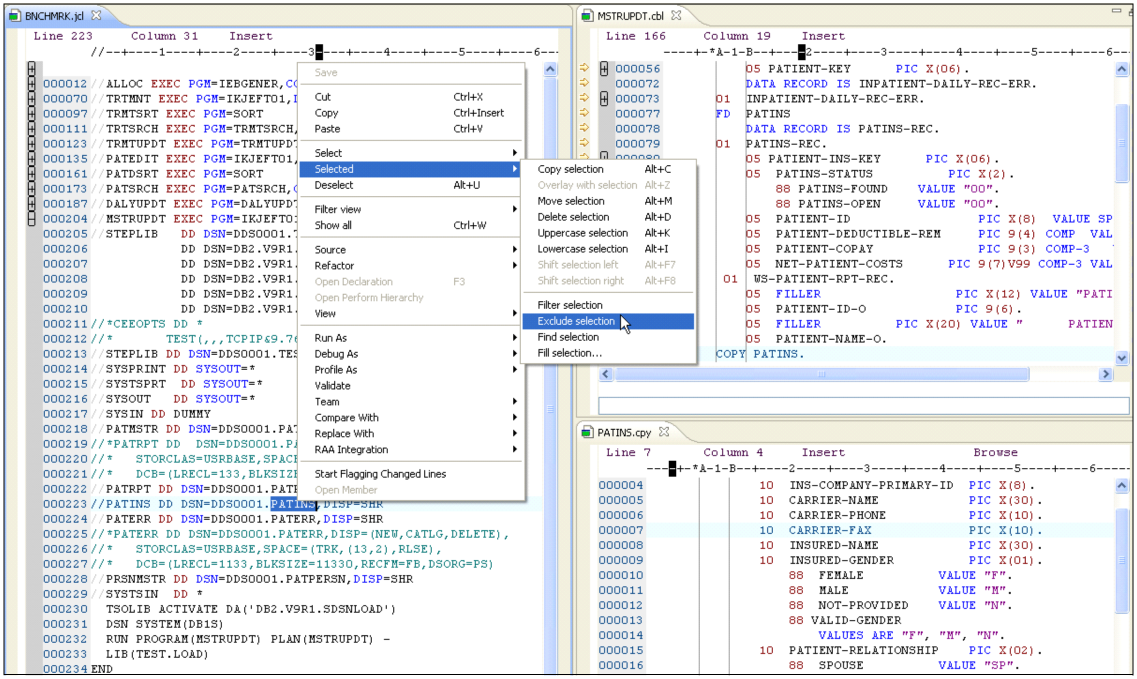


Figure 4-21 Editing a JCL batch job that is filtered by EXEC statements, excluding all references to a VSAM file, and viewing a program and a VSAM copybook for the VSAM file

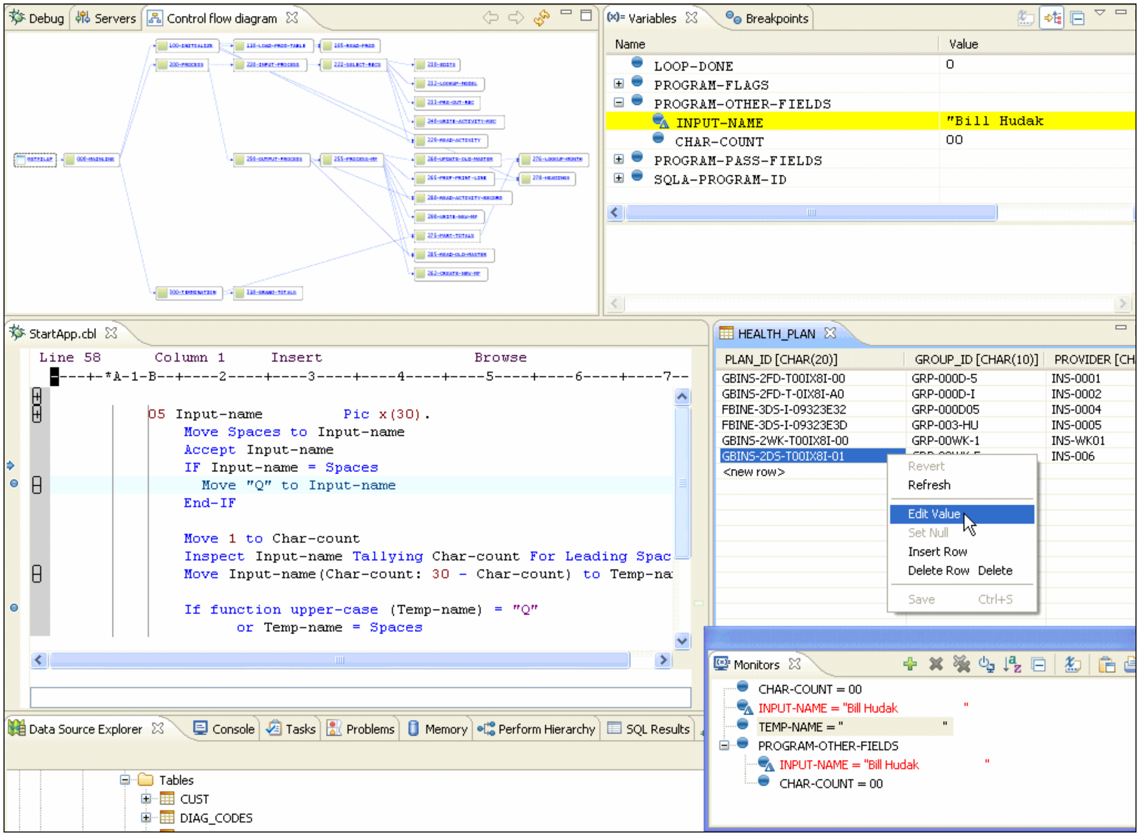


Figure 4-22 Debugging a COBOL program, showing its control flow, variable values, and monitored (selective variable values), and modifying a DB2 test table during debugging

4.3.7 Context menu options

The LPEX editor provides a context menu (Figure 4-23) that, along with the various tools from the main (or top) menu in the workbench, provides shortcut access to the following tasks:

- With source code editing options, you can cut, copy, and paste (*upper* and *lower* case), including working with code blocks versus line and character editing mode.
- Filtering for optimal viewing of large programs is provided. In COBOL, you can filter the selected text, divisions, an outline view showing 01-fields, paragraphs, and sections, selected keywords or text, comments and code, SQL, CICS, and Data Language/Interface (DL/I) statements, syntax errors, and more.
- Variable, paragraph, and section name refactoring gives you a visual change of all dialog, showing a preview of what will be affected in your source before you initiate the actual changes.
- If you have selected a paragraph or section as opposed to a variable, you can see a code sequence diagram, which is language-dependent. Figure 4-24 on page 122 shows the COBOL perform hierarchy. This feature, which is built dynamically during your edit session so that it reflects changes to your code immediately, is extremely useful for understanding program control flow and how your program executes.
- Open declaration, which navigates directly to the variable or paragraph declaration in your program, allows for the use of tool bar, icon, or arrow keys to jump back to the current edit position.
- You can open (for edit) or browse (read-only view) copybooks and includes.
- Syntax checking of your program can be done locally on the desktop.
- You can do a compare with the previous edited version of your source file or a history version of your file. This capability shows your changes visually and provides a way out of inadvertent mistakes.
- Similar to compare, a “replace with” function allows you to recover to a previously saved state of your program or source file.
- You can work synchronously with your SCM product.

Because the context menu provides the features within the context of what you are doing during your development, it is extremely easy to use, intuitive, and powerful (Figure 4-23 and Figure 4-24). You can enhance the context menu with custom extensions (see the sections on Menu Manager: 4.8, “Host integration with Menu Manager” on page 176 and Rational Asset Analyzer 4.7, “Integration with Rational Asset Analyzer” on page 160.

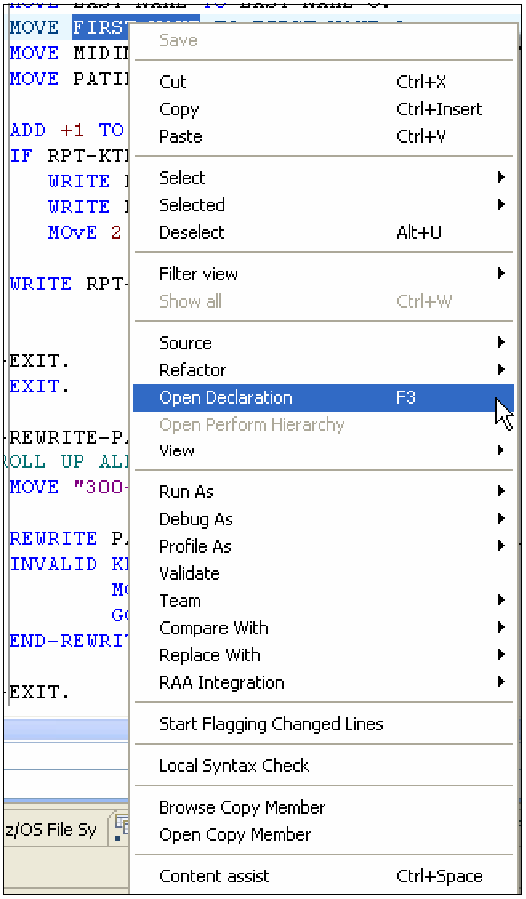


Figure 4-23 LPEX editor’s context menu

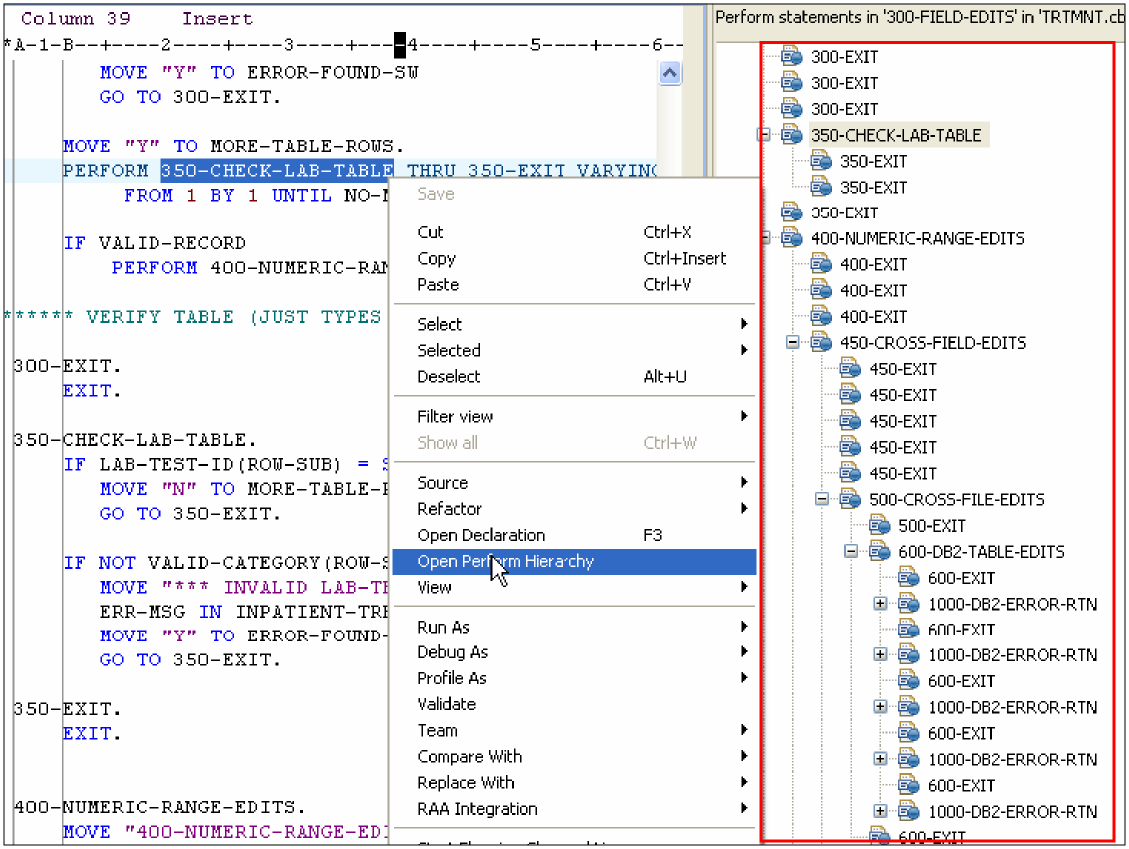


Figure 4-24 Context menu: Perform hierarchy

4.3.8 ISPF editing options

For those veteran ISPF developers with established ISPF editing techniques, the Rational Developer editor provides an ISPF profile option that activates these familiar capabilities (Figure 4-25):

- Prefix area for ISPF prefix commands: Nearly all prefix area commands are available and compatible.
- Command line for ISPF line commands: The majority of the editing line commands are available and compatible.

Along with the ISPF profile, you can also change the background colors of the editor by selecting **Window** → **Preferences** → **LPEX Editor** → **Appearance** and **Parsers** → **Parser Styles** to achieve a mainframe-like ISPF look and feel during edit (that is, aqua or green text on a black background).

And finally, several documents have been produced that describe the exact ISPF/Rational Developer compatibility, edit-option for edit-option. On the COBOL Cafe, you can find a comparison PDF from the Soforte' consulting group in Europe at the following website:

http://www.ibm.com/developerworks/rational/cafe/docBodyAttachments/3359-102-1-6944/Equivalence_ISPF_and_LPEX_Editor_Functions_V220.pdf

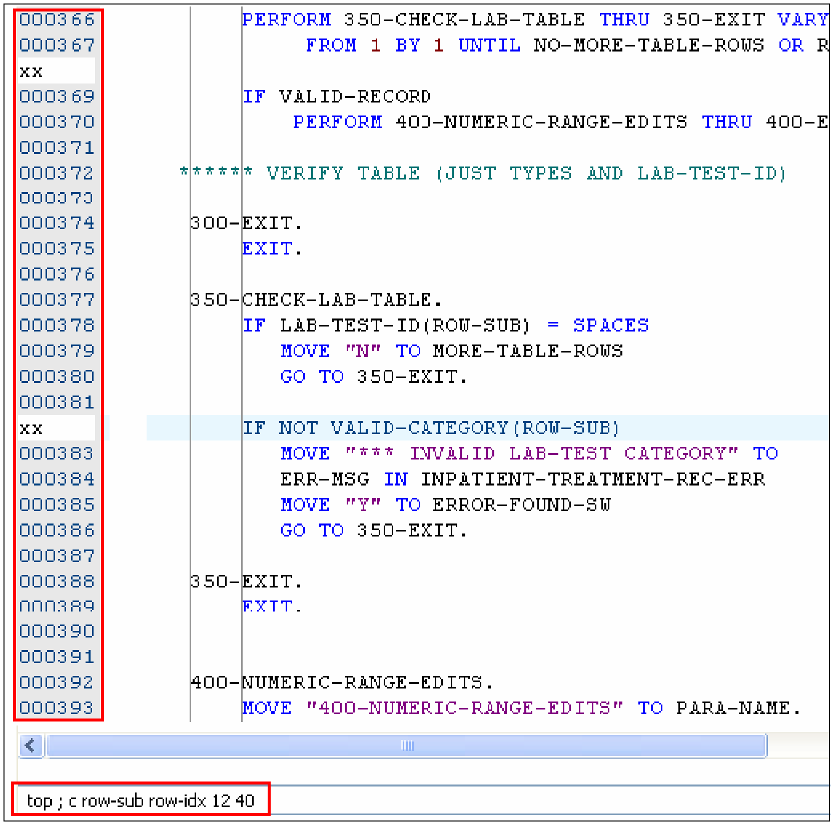


Figure 4-25 ISPF editing: Command line and prefix area

4.3.9 Compiling and syntax checking

There are two reasons for compiling programs:

- Shake out syntax errors from the source
- Produce the binaries (.OBJ and load modules) that are used in application runtime execution

Several Rational Developer options can help you reduce the number of syntax errors in your code:

- Content Assist: By pressing Ctrl+spacebar, a list of legitimate options for source inclusion appears in a pop-up window. You select an option, instead of having to type it. Therefore, you have fewer typing mistakes.
- Validation: As part of its language-sensitive nature, LPEX validates your syntax after each key stroke. Syntax validation errors show up as a small yellow triangle warning sign in the left border (Figure 4-26). By hovering your mouse over the warning sign, you can see the syntax error message for the statement. All the languages (COBOL, PL/I, HLASM, JCL, SQL, program specification block (PSB)/database description (DBD) statements, CICS statements, MFS, and BMS) support validation checking.
- Local syntax checking for PL/I and COBOL (including DB2 and CICS embedded application programming interfaces (APIs)): The context menu provides a Local Syntax Checking option that invokes the Windows compilers associated with your program source on your desktop computer. This process is fast, convenient, and saves mainframe CPU resources that otherwise are used for compiling your programs through JCL or in the TSO foreground.
- Remote syntax checking: For all supported languages, Rational Developer uses a set of properties (see 4.3.10, “Rational Developer property groups” on page 127) to start background JCL jobs dynamically that perform a number of functions, including remote syntax checking using the compile PROCs you specify (either a set of IBM-supplied procedures or your own PROCs).
- Generating JCL for compiling: Using the same property groups, Rational Developer can generate JCL for you to submit and follow using Remote Systems Explorer for compiling and linking.

For local syntax checking and remote syntax checking (either through the submitted or generated JCL), a Remote Error list view (Figure 4-27) is populated with syntax error messages that are hyperlinked back to the originating source statement and, in certain cases, actually to the character position in the line where the error was detected. This capability makes finding and fixing syntax errors a simple matter of point-and-click with your mouse to get to the error statement and to use the message to fix your syntax problem.

For creating the binaries, Rational Developer (using properties that you customize in property groups) link-edits successfully compiled binaries to the appropriate .OBJ and load libraries.

Additionally, you have these capabilities:

- As of this writing, you can use one of these compilers for local syntax checking:
  - Enterprise COBOL 4.1
  - Enterprise PL/I 3.9
- For BMS and MFS, you can specify the appropriate Assembler procedures and online libraries to link the format and BMS.
- For IMS, you can specify the appropriate resource library (RESLIB) to concatenate into the JCL.
- For CICS, you can specify the CICS libraries into which to promote the programs.

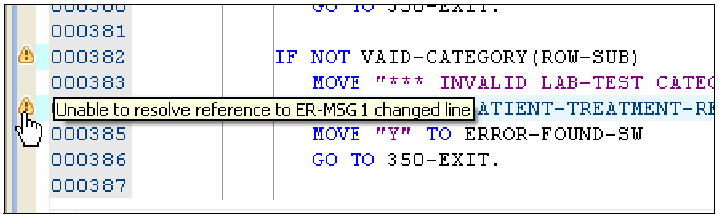


Figure 4-26   LPEX editor: Dynamic syntax validation

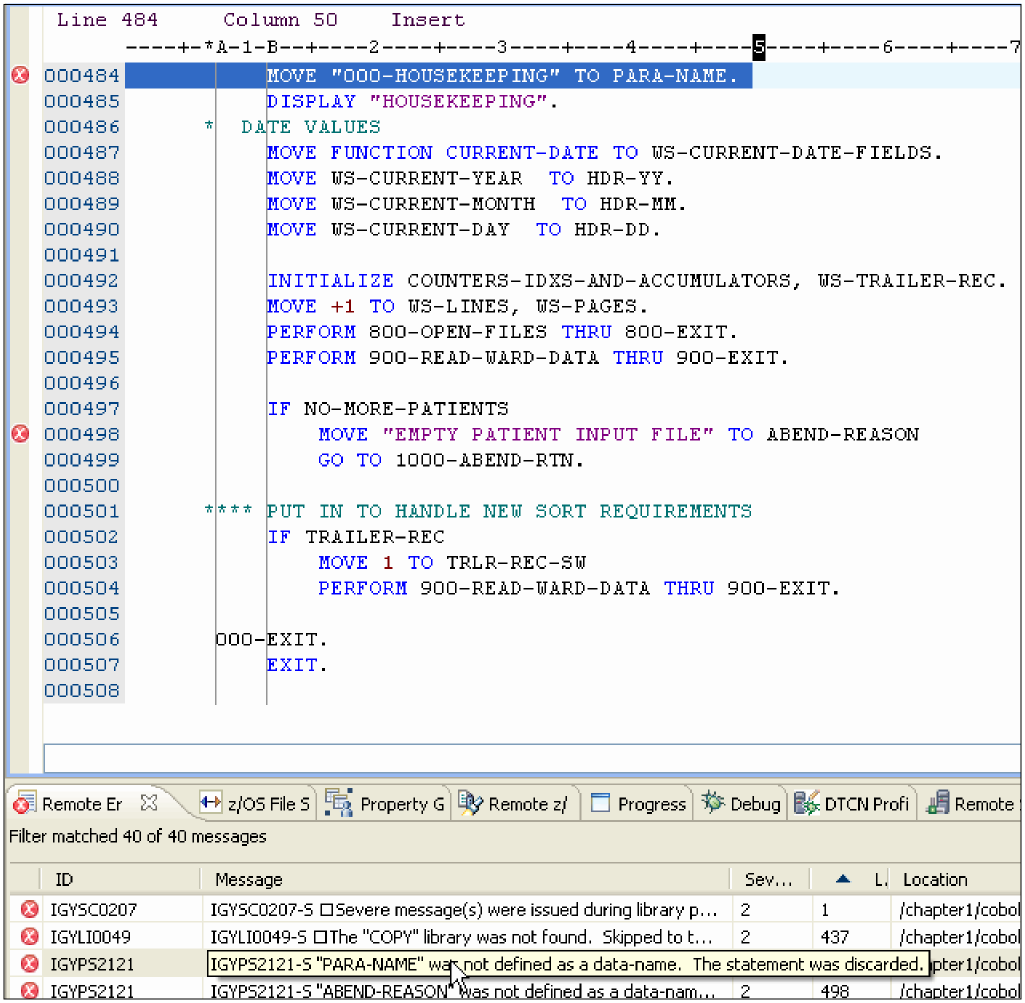


Figure 4-27 Integrated hyperlinked syntax error messages

4.3.10 Rational Developer property groups

In order for Rational Developer to manage the local and remote syntax check and compilation process previously described, as well as to provide key editing features, such as browse or edit copybooks, you specify entries in your workspace's property groups.

Property groups are XML files where you customize details about z/OS PROCs and data sets, libraries, compile, link, DB2 bind, BMS/MFS, and HLASM assemble options. Rational Developer has a Property Groups Manager (Figure 4-28), which allows you to edit a property group file and customize the following characteristics:

- Categories of language and runtime resources that you want to support for your workspace
- For each selected category, specify the following details (Figure 4-28):
  - Runtime environments, such as DB2, CICS, or IMS (any one, two, or all three environments in combination)
  - Procedures and steps for various combinations of DB2 or CICS preprocessing, compile, link, assemble options, and data sets and libraries
  - Local (workstation) compiler options
  - The ability to invoke a local custom preprocessor (not the DB2 or CICS preprocessing step in the standard compilation process)
  - JCL substitution, a facility that allows you to set up parameters that are substituted at run time, simplifying the number of run time, proc, and property groups that you have to develop and maintain

After you define or customize a property group, you associate the property group with a resource in your workspace, such as a library, a filter (a number of libraries defined as an organizational unit), a local project or MVS subproject, or even a single data set.

Because the nature of mainframe computing resources is both in-depth and broad with a large number of potential technology combinations and because Rational Developer's support for your functionality is comprehensive, the IBM z/OS environment is extensive.

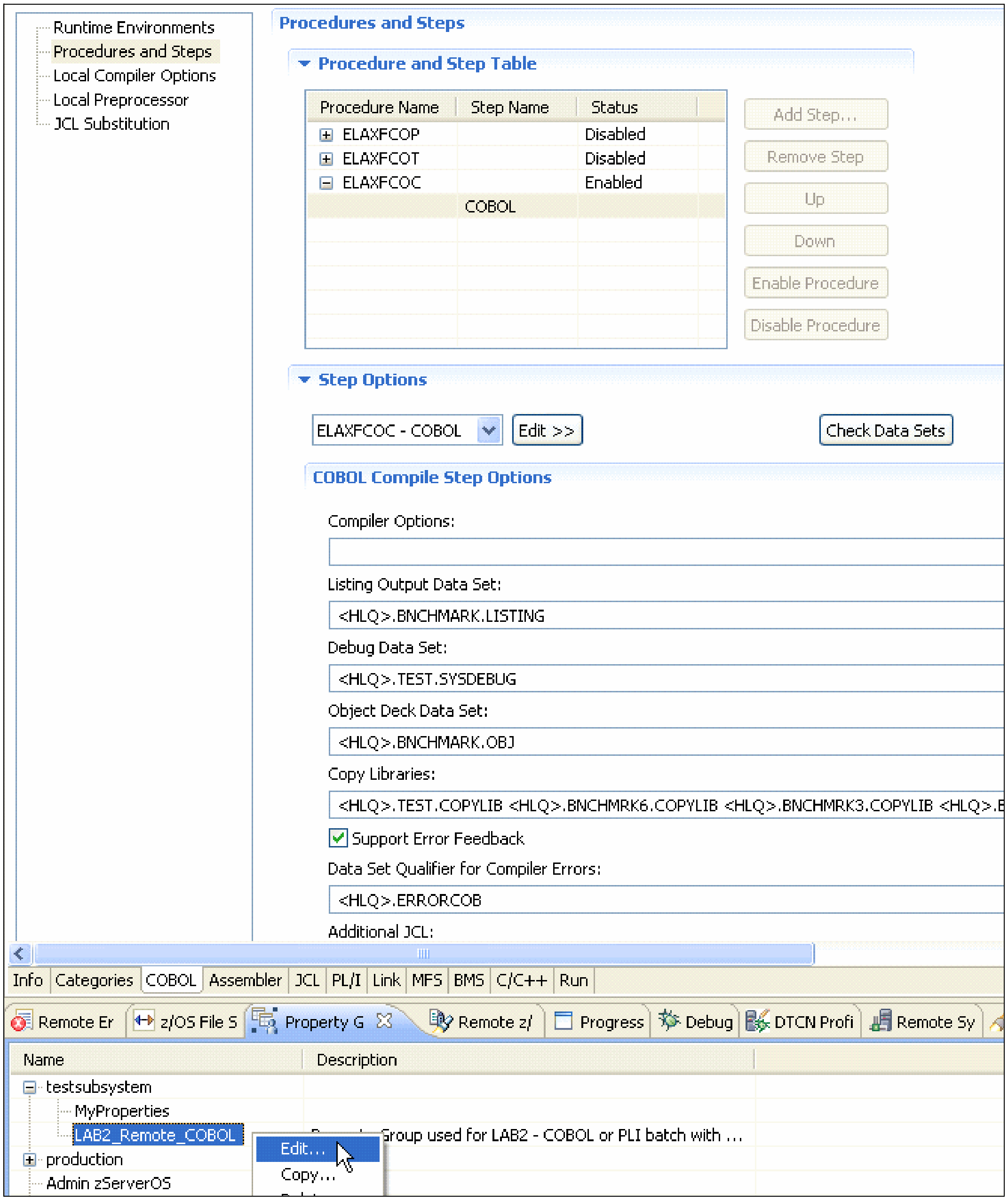


Figure 4-28 Rational Developer property file for COBOL compile

4.3.11 Code templates and snippets: Code stereotypes

Rational Developer ships with several features that support code reuse:

- Code templates
- Code snippets

Code templates

Code templates (Figure 4-29) are skeleton program files that you can customize, export, and roll out to developers from the Rational Developer workspace defaults. At this release, Code Templates are only available for COBOL programs. Rational Developer uses these skeleton files when you create a new program from the menu by selecting **File** → **New** → **Program**.

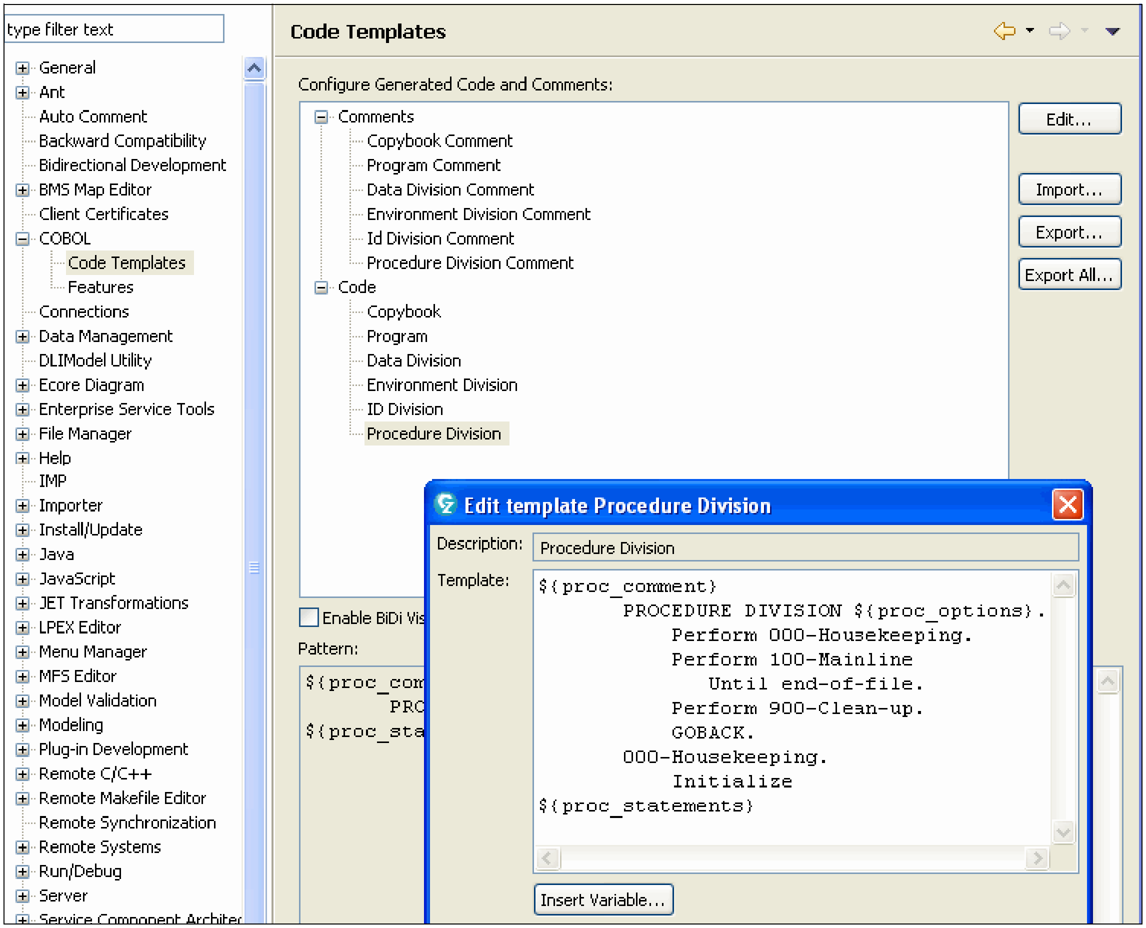


Figure 4-29 Customizing a code template

Figure 4-29 on page 129 shows a number of code template areas that you can customize for these functions:

- Jump-start program building from a skeleton that includes specific comments, variables, and code
- Institutionalize new development to conform with shop standards and best practices

When you create new programs, you can use additional templating features for CICS and DB2 programs that allow you to pick up standard EXEC statements and so forth for CICS and DB2 programs.

Code snippets

Code snippets are a flexible and frequently used tool for code reuse (Figure 4-30). A *snippet* is any code from a single keyword, which is not really that useful, to statements, paragraphs, sections, and entire programs, organized in patterns or drawers inside the snippets palette. Snippets are available for insertion in programs (all languages), JCL, SQL, MFS, and data files.

Rational Developer ships with a default set of snippets. You can obtain additional snippets for the code that is used in this book at this website:

http://www.ibm.com/Redbooks/SG247868/code

Code snippets allow for variable substitution. For example, OPEN ${*FileName*} prompts you to enter the DATA DIVISION file name when you use the snippet. Variable substitution makes snippets flexible and easily repurposed.

Perform these steps to create a code snippet:

1. Select and copy code inside the editor
2. Right-click over the Snippets view and select **Paste as Snippet**.

Perform these steps to use a code snippet:

1. Set your mouse cursor focus inside the editor to the position into which you want the snippet copied.
2. Open the appropriate drawer in the Snippets view (Figure 4-30).
3. Double-click the snippet that contains the code that you want dropped into the file. Enter any variable value substitutions that are defined for the snippet.

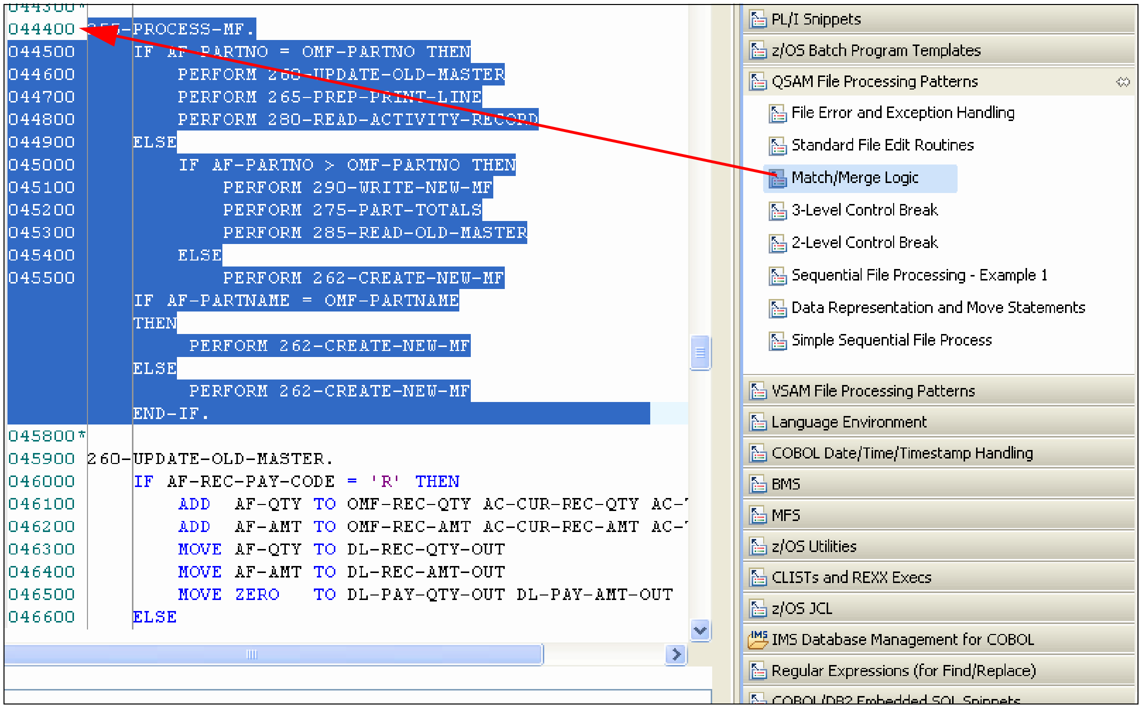


Figure 4-30 Custom snippet categories

4.4 Rational Developer language and runtime support

Rational Developer supports the development of COBOL, PL/I, HLASM, Java, Enterprise Generation Language (EGL), and C++ applications for the following run times:

- Batch
- DB2
- IMS Transaction Manager (IMS TM) systems
- CICS
- Microsoft Windows
- IBM AIX® using Rational Developer for IBM System p®

Next, we describe these languages and run times in-depth.

4.4.1 COBOL

As of this writing, Rational Developer for System z supports Enterprise COBOL Version 4.1 at the tooling, compilation, and runtime levels. You can develop OS/VS COBOL, VS COBOL II, IBM S/390® COBOL, and Enterprise COBOL programs using Rational Developer for System z without change using the Enterprise COBOL compiler, unless you have one or more of the following types of programs:

- Programs that are compiled with the CMPR2 compiler option
- Programs that have System Object Model (SOM)-based object-oriented COBOL syntax
- Programs that are compiled for OS/VS COBOL, VS COBOL II, or S/390 COBOL that use words that are reserved in Enterprise COBOL now
- Programs that are *deprecated* in Enterprise COBOL, such as TRANSFORM
- Programs that have undocumented IBM COBOL extensions

Rational Developer for System z has a high degree of compatibility with and functionality for IBM COBOL solutions across all product features.

4.4.2 PL/I

As of this writing, Rational Developer for System z supports Enterprise PL/I V3.9 at both the tooling and compilation levels. You can develop OS PL/I, PL/I for MVS and VM, and Enterprise PL/I programs using Rational Developer for System z, without change using the Enterprise PL/I compiler with some restrictions (you can consult your IBM technical representative for a detailed list).

PL/I offers many development options:

- Syntax highlighting, including the operands required for the opcode
- Content Assist for PL/I:
  - Statements
  - Variables
  - Procedures
- Virtually all the same views and analysis and editing capabilities in LPEX for COBOL are available for PL/I:
  - Program Outline view
  - Source Properties view
  - Bookmarks and tasks
  - Local and remote syntax checking

– Figure 4-31 shows the LPEX editor's context menu, code filtering, split screen, language colorization, source compare, history files (for version deltas), validations, and so forth

- F1 Help links to the PL/I language reference manual
- Integrated build capabilities
- Symbolic debugging of language environment-enabled PL/I
- Symbolic debugging of non-language environment-enabled PL/I
- Ability to debug from PL/I to COBOL, HLASM, PL/I, Java, and return. While debugging, you have the ability to alter registers, storage, and variables and to add statements.

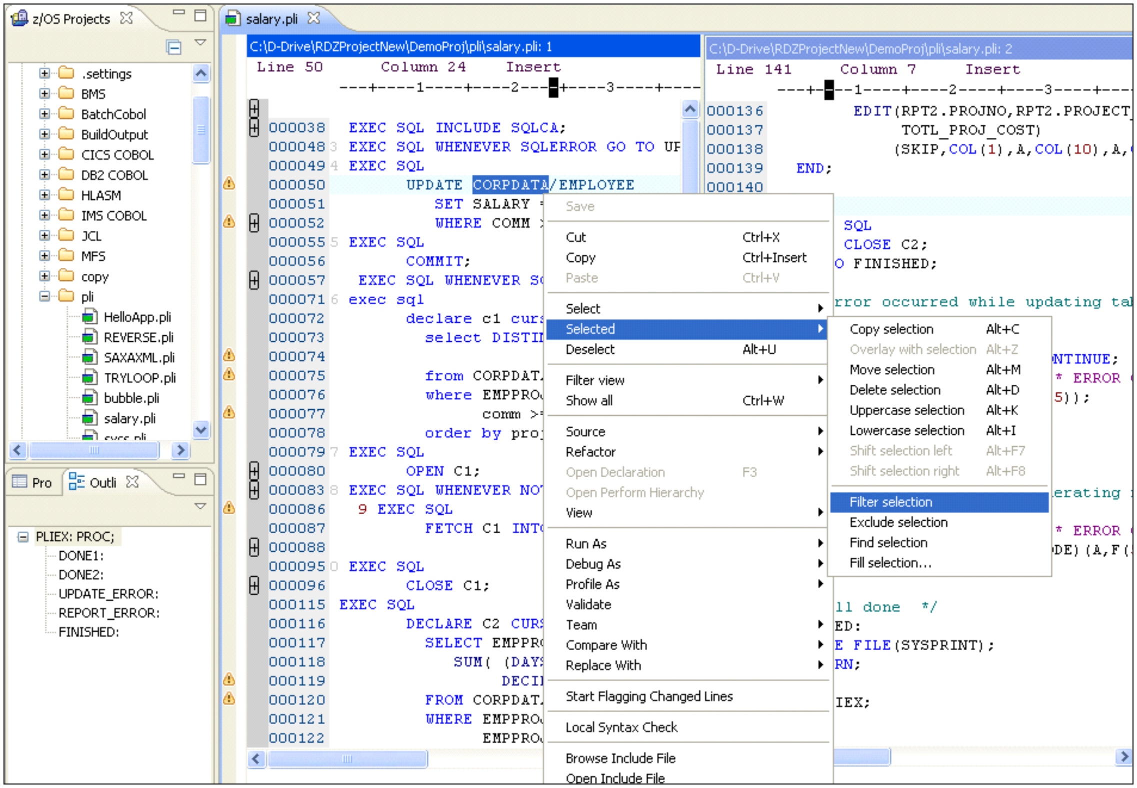


Figure 4-31 Developing PL/I code using Rational Developer for System z: LPEX editor features

4.4.3 HLASM

As of this writing, Rational Developer for System z supports High Level Assembler (HLASM) programming at both the tooling and compilation levels. You can develop HLASM programs using Rational Developer for System z, without change.

HLASM offers the following development options, among others:

- Syntax highlighting, including the operands required for the opcode
- Content Assist for HLASM
- Virtually all of the same views and analysis and editing capabilities that are available in LPEX for COBOL are available for HLASM also:
  - Program Outline view that organizes your labels by sections, symbols, and macros and by using domains
  - Context menu/filter allows for consolidated or isolated views of Assembler instructions, branch instructions and labels, macros, Transaction Processing Facility (TPF) macros, and TPF Database Facility (TPFDF) macros
  - Source Properties view
  - Bookmarks and tasks
  - Syntax checking

    See Figure 4-32 showing the LPEX editor's context menu, code filtering, split screen, language colorization, source compare, history files (for version deltas), syntax validations, and so forth
- F1 Help links to the Principle of Operation manual, describing Assembler instruction (opcode and operands)
- Integrated build capabilities
- Symbolic debugging of language environment-enabled HLASM
- Symbolic debugging of non-language environment-enabled HLASM
- Ability to debug from a high-level language (HLL), such as COBOL or PL/I, to HLASM and return. While debugging, you have the ability to alter registers, storage, and variables and to de-reference pointers.

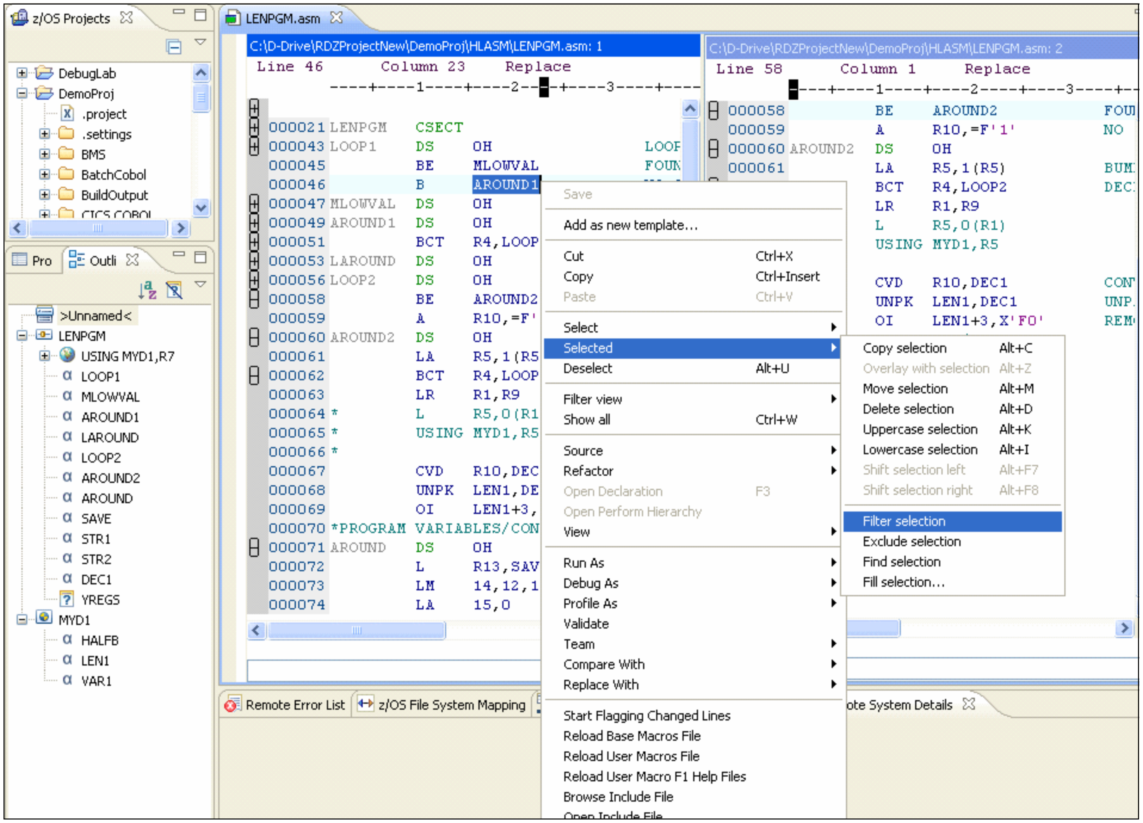


Figure 4-32   Developing HLASM code using Rational Developer for System z: LPEX editor features

4.4.4 DB2 and Rational Developer for System z

Rational Developer for System z provides many high-value tools and productivity features for performing SQL work with DB2 in COBOL, PL/I, HLASM, Java, EGL, and C/C++:

- Embedded SQL statement validation:
  - Keyword syntax
  - DB2 object semantic validation and table/view/column catalog lookup:
    - Remote syntax check (default)
    - Local syntax check, which requires a local copy of DB2 UDB and, of course, the Data Definition Language (DDL) for the DB2 objects generated under DB2 UDB. If you use local syntax checking for SQL, you potentially reduce your z/OS DB2 BIND operations' time and cost.

  - Interactive (dynamic) SQL GUI tools and Content Assist-based development
  - Stored procedures' wizard-driven development
- DCLGEN: Creating copybooks from table schemas for COBOL, PL/I, C, and Java, with the Java Developer's Toolkit V3.1 or later
- A wizard for building, testing, and deploying DB2 stored procedures:
  - Native SQL stored procedures
  - COBOL stored procedures
- SQL statement creation and testing:
  - Using Content Assist
  - Using a graphical SQL statement creation facility (Figure 4-34 on page 139)
  - Using snippets (see 4.3.11, "Code templates and snippets: Code stereotypes" on page 129)
- DB2 entity relationship data analysis (the overview diagram)
- DB2 data object DDL generation and compare for these objects:
  - Tables
  - Indexes
  - Views
  - Synonyms and aliases
  - Stored procedures
- Test table data management

  See Figure 4-33 on page 138, which shows table editing, but it does not show these other options:
  - Table unload and table load operations: Combining table unload and table load operations with the multiple connections that are allowed makes it incredibly easy to take copies of data
  - Viewing sample values
- SQL statement performance analysis using these features:
  - A simple-to-use relative query cost feature that shows the relative time expended to execute an SQL statement
  - DB2 EXPLAIN facility:
    - Through integration with Visual Explain, which is a no-charge plug-in to Eclipse

- The standard EXPLAIN tables and the process of analyzing a statement's physical execution path (indexed versus scanned, which index, and so forth)

Debug tools

In addition to these Data perspective tools, Rational Developer for System z also provides developers with the following tools:

- The ability to debug (using IBM Debug Tool) DB2 stored procedures running on z/OS
- The ability to debug (using IBM Debug Tool) batch and online COBOL, PL/I, HLASM, Java, and C/C++ with embedded SQL
- DB2 (embedded SQL) COBOL program templates as features for new programs (see 4.3.11, "Code templates and snippets: Code stereotypes" on page 129)

In the preceding list, you integrate the SQL statement validation into the LPEX editor. The remaining features are all part of the Data perspective, which you access by selecting **Window** → **Open Perspective** → **Other** → **Data**.

The Data perspective

The Data perspective allows you to connect to a DB2 database instance on any IBM system that you can access by using a Java Database Connectivity (JDBC) driver and by entering the proper values for the following variables:

- Location: External name of the DB2 subsystem
- Host: IP address of the z/OS running your DB2 instance
- Port number: Port number of the listener for incoming JDBC access
- User ID and password: Credentials that provide authorized access to the DB2 subsystem and all data objects

After you connect, you can perform all the work listed by using rich graphical tools, as opposed to the arduous process of accessing SQL Processor Using File Input (SPUFI) or IBM Query Management Facility™ (QMF™) and coding and executing SQL statements to manipulate test data.

Figure 4-33 shows an example Data perspective session:

- The Data Source Explorer (lower left pane) lists the DB2 connection and the available schemas (filtered for ease of use) of the types of DB2 objects with which you can work.
- The EMP table, which comes with DB2 as a sample table on most platforms, has been selected. You can see its columns, indexes, and dependencies (foreign key values to a parent table if one exists) and more metadata about the DB2 objects.

- Figure 4-33 shows editing three tables (Ward_Data, Hosp_Position, and Emp) in full screen edit mode, which is similar to manipulating values in Excel:
  – Modifying existing row values by simply typing
  – Adding new rows
  – Deleting existing rows

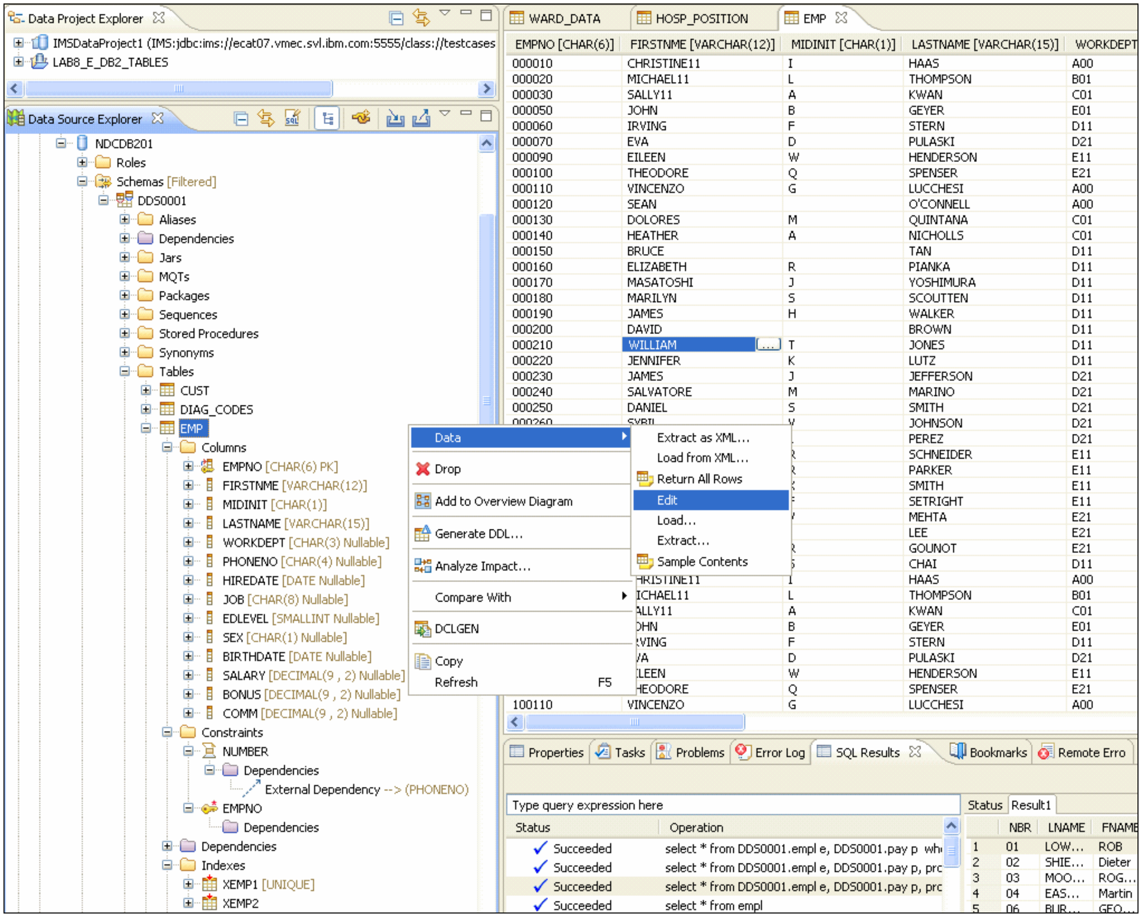


Figure 4-33 The Data perspective: Working with DB2 table data

Because the Data perspective accesses DB2 objects through a JDBC driver, this feature of Rational Developer can save considerable development LPAR host resources compared to using QMF or SPUFI.

There are two options for creating, syntax checking, and running (testing) SQL:

- Figure 4-34 shows a graphical SQL statement builder.
- You also can use Content Assist (Ctrl+spacebar). Similar to the LPEX COBOL editor, PL/I statement building functionality allows you to select the schema and table names and build out your SQL at the language level. This function is useful for developers with more SQL experience.

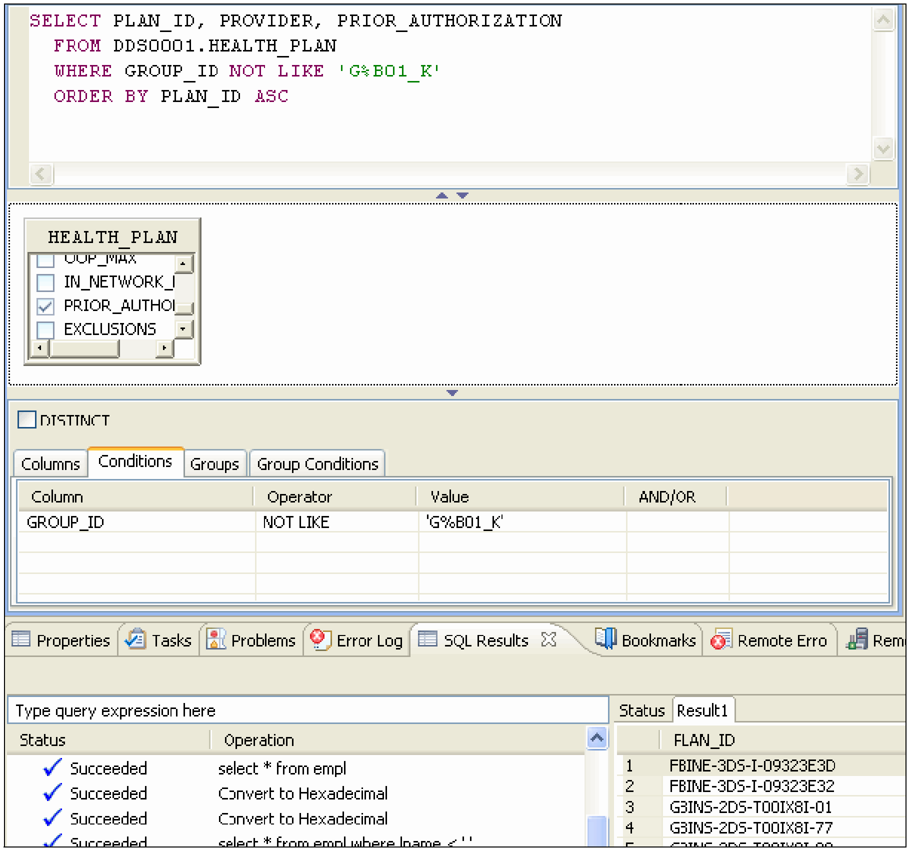


Figure 4-34 SQL statement builder and SQL results

The Data perspective is one of the most powerful facilities in Rational Developer. Developers, test management teams, and database administrators (DBAs) all take advantage of this useful tool.

4.4.5 IMS and Rational Developer for System z

Although arguably not as robust as the DB2 tooling, Rational Developer for IMS provides a similar set of high-value functions for working with IMS applications, MFS screens, and DL/I databases, including but not limited to these functions:

- Embedded IMS statements in COBOL, PL/I, HLASM, C/C++, and Java
- IMS database segment entity relationship data analysis using the overview diagram in the Universal Driver and the DL/I database model plug-in
- The ability to debug (using IBM Debug Tool) IMS batch (DL/I and batch message processing (BMP) program) and IMS online (message processing program (MPP) and query BMP Queued Batch Message Processing (QBMP) program) applications that are written in COBOL, PL/I, HLASM, and Java
- The ability to submit a batch for business transaction services (BTS) application testing and DFSDDLT0 DL/I statement testing
- An MFS 3270 editor (Figure 4-35), which allows you to develop or maintain and support MFS files graphically and to use LPEX source editing

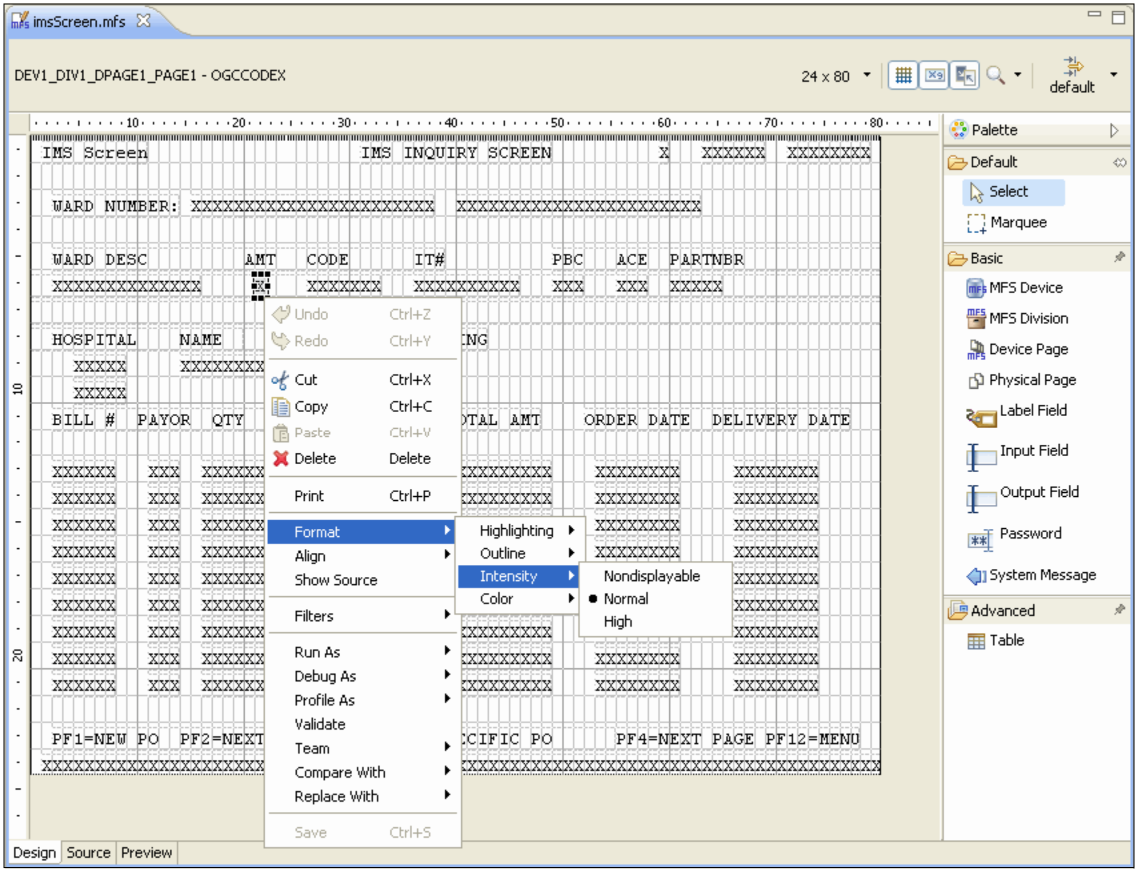


Figure 4-35 Rational Developer MFS screen editor

- A DL/I database model Eclipse plug-in, which shows the database hierarchy graphically (Figure 4-36). Note that this facility, which is enabled from an IMS perspective, allows you to view, document, and understand these aspects of your IMS database:
  - DL/I hierarchy
  - Senseg fields through the PSB/DBD
  - COBOL copybook segment overlay fields that are logically defined in a COBOL or PL/I copybook but that are not defined in the PSB

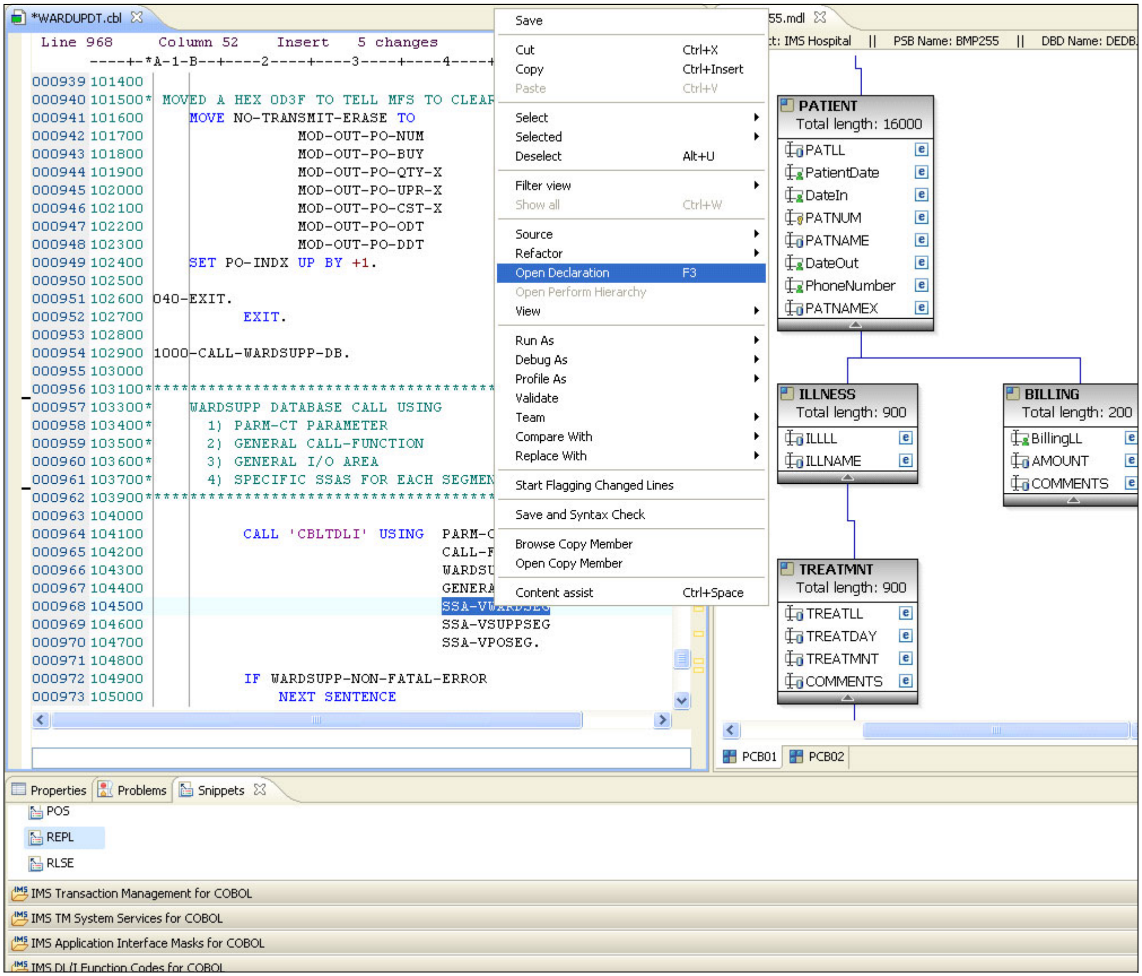


Figure 4-36 Rational Developer and IMS application development tools

- The ability to access IMS databases directly through the IMS Universal Driver (Figure 4-37) to perform these tasks:
  - View segment and PSB senseg field metadata
  - Browse segment data values, which is often difficult, especially with complex hierarchical structures
  - Search through IMS databases

  During application analysis, you often need to verify your understanding of values within an existing database, and the Universal Driver provides a simple, SQL-driven process for this verification.

– Edit and update segment data
– Return statistics on segment value cardinality (the number of discrete values in all the segments throughout your IMS database).
– Unload a segment to a flat file
– Code and test segment access strategies

- Test database segment data management (Figure 4-37 shows table editing, but it does not show many options, such as load and unload or viewing sample values)

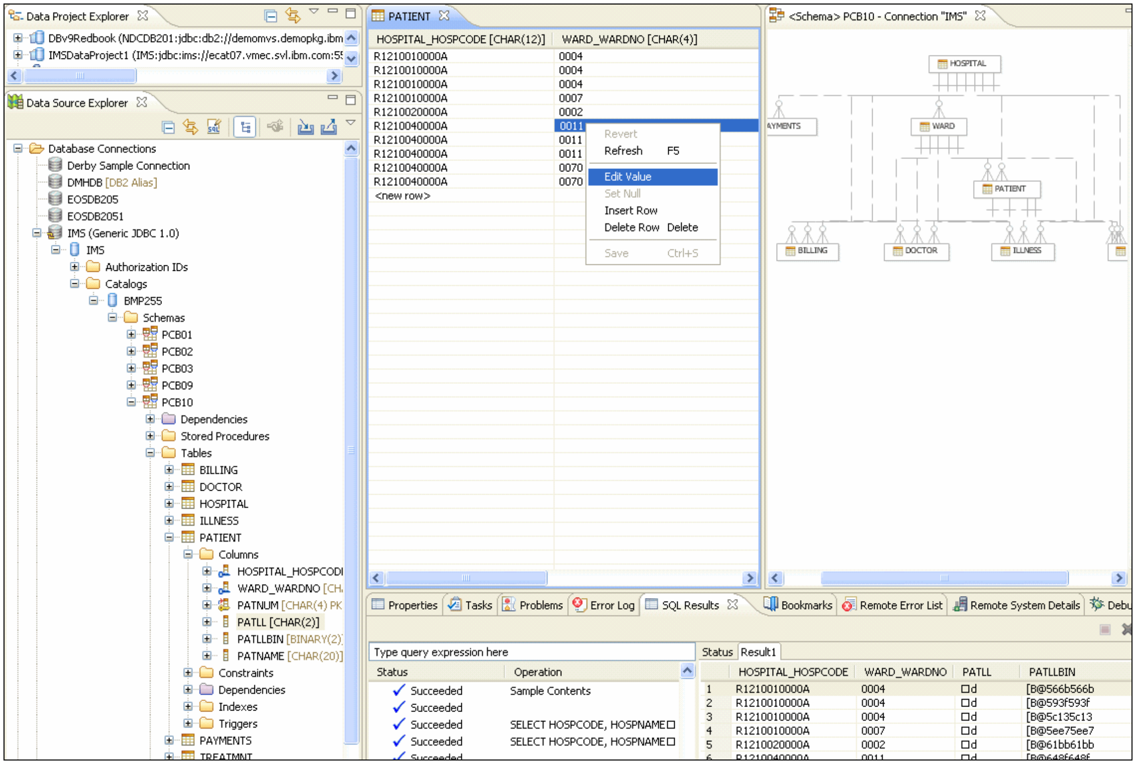


Figure 4-37 Rational Developer IMS database access through the Universal Driver showing segment data being edited, segment senseg/fields and properties, the results of a read access, and the DL/I hierarchy

- DL/I statement creation and testing:
  – Using Content Assist
  – Using a graphical SQL statement creation facility (Figure 4-38)
  – Using snippets

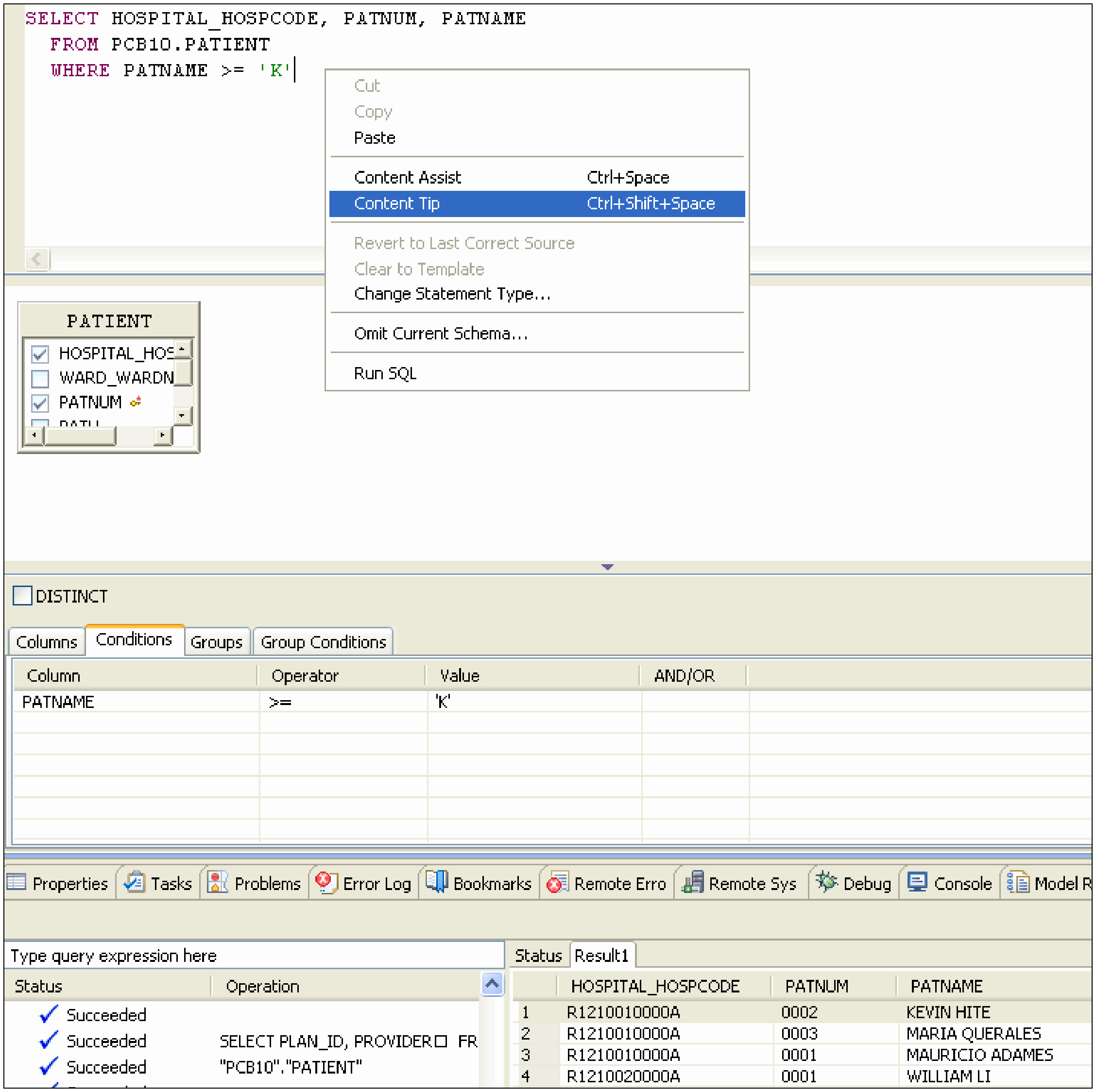


Figure 4-38 IMS DL/I model-driven statement creation

In the preceding list, the IMS statement validation is integrated into the LPEX editor. The remaining features are all part of the Data perspective, which you access by selecting **Window** → **Open Perspective** → **Other** → **Data**.

4.4.6 CICS and Rational Developer for System z

Similar to its transaction management support for IMS, Rational Developer provides a set of high-value functions for working with CICS applications, BMS screens, and CICS-DL/I statements, including but not limited to these functions:

- Embedded CICS statements in COBOL, PL/I, HLASM, C/C++, and Java

  EXEC CICS statement:

  – Validation

  – Remote syntax check (default)

  – Local syntax check, which requires a local copy of the IBM TX Series for Multiplatforms. If you use local syntax checking for CICS statements, you potentially reduce your z/OS CICS preprocessing time and costs.

  – CICS DL/I statements

  – Content Assist, which is used to develop command-level CICS statements

- A CICS BMS map editor (Figure 4-40 on page 148), which allows you to develop or maintain and support BMS files graphically and to use LPEX source editing functionality against your Macro statements
- Integration with the CICS Explorer. CICS Explorer allows you to manage connections, define CICS resources, and perform other CICS SysAdmin tasks.
- The ability to debug CICS online applications that are written in COBOL, PL/I, HLASM, and Java using one of these tools:

  – IBM Debug Tool, in which case, the debugging occurs under z/OS
  – TX Series, in which case, the debugging occurs on a workstation

- CICS (EXEC CICS) COBOL program templates, as features for new programs (see 4.3.11, “Code templates and snippets: Code stereotypes” on page 129)

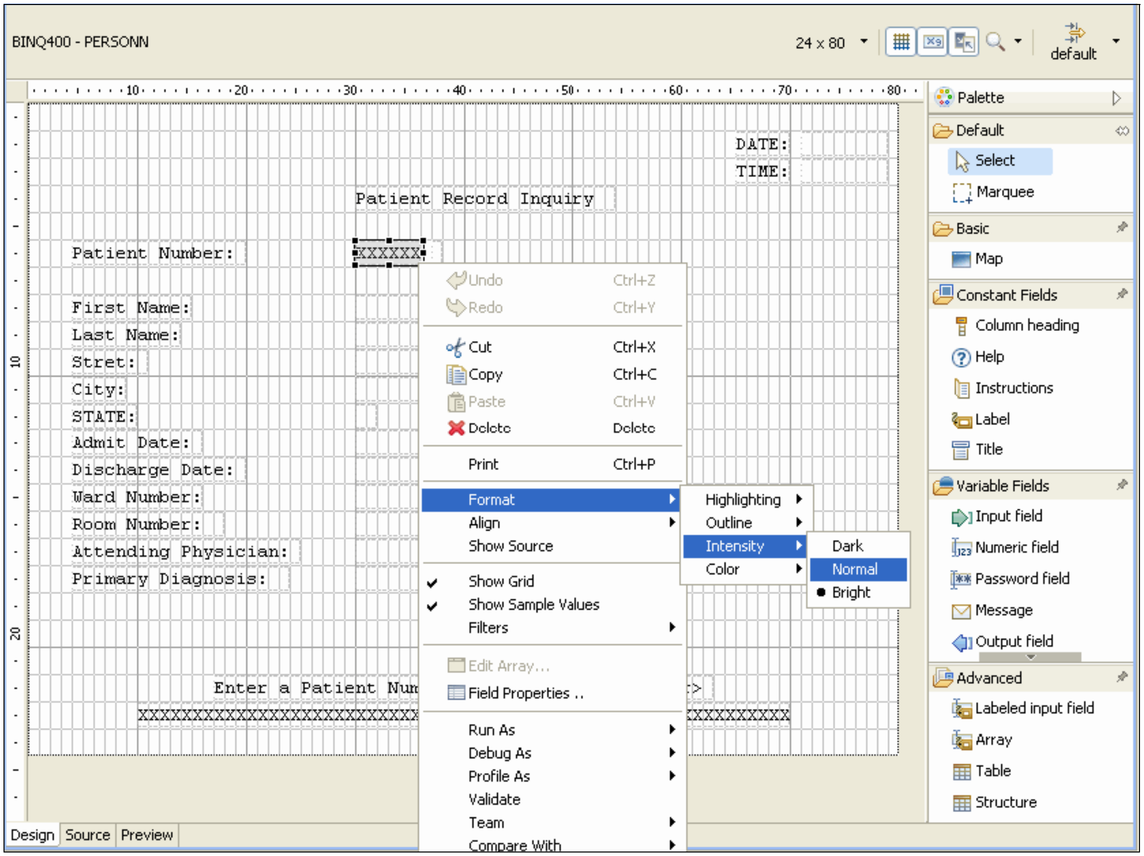


Figure 4-39 The CICS BMS map editor

4.4.7 Rational Developer JCL and batch processing

Rational Developer provides a set of high-value functions for working with standard z/OS batch JCL and utilities, and for submitting and managing batch jobs through the Job Entry Subsystem (see 4.3.2, "Remote Systems Explorer and batch job management" on page 107).

Rational Developer allows you to work with JCL and batch processing in the following ways:

- The LPEX editor supports JCL to allow you to perform these tasks:
  - View JCL in the same color-coded manner as COBOL, PL/I, and HLASM
  - Code new JCL statements:
    - Standard (manual) coding process
    - Using JCL snippets
  - Validate the statements for JCL language syntax
  - Use most of the context menu options for editing and viewing:
    - Filter by JCL elements (Figure 4-40 on page 148)
    - Select and exclude or unexclude text
  - Submit the JCL from inside an edit using an LPEX command (`sub` just as in ISPF) or from the context menu

You can submit jobs to the default connection in which you are working (if you are logged in to your z/OS through Remote Systems Explorer) or to a specific connection (see Figure 4-41 on page 150, which shows the Submit option to the user's *production* connection).

Rational Developer has several features that generate JCL for these functions:

- Compile
- Compile and link
- Compile, link, and go

You can capture the JCL that is generated for compile and link or go and save it into a PDS for reuse. You can submit JCL as a menu action by using the Menu Manager.

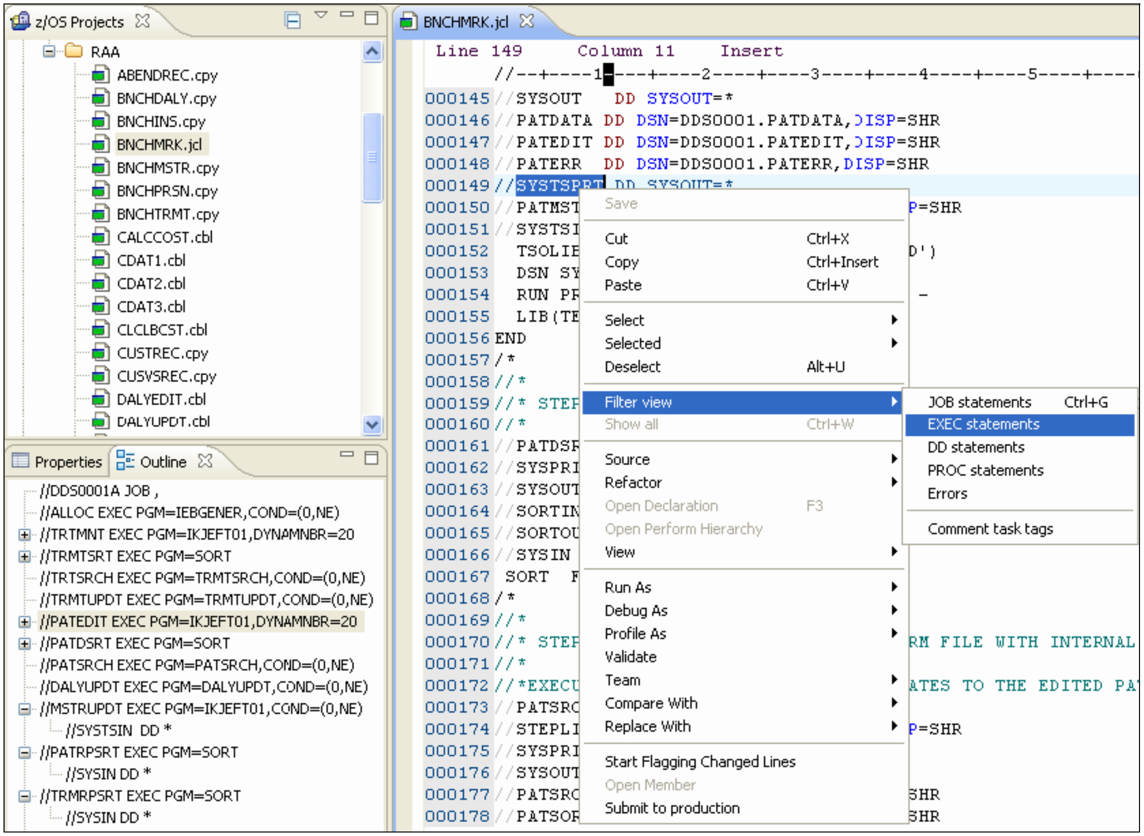


Figure 4-40 Editing JCL in LPEX

4.5 Rational Developer and source control management integration through CARMA

Rational Developer integrates with all of the major source control management products and systems, including but not limited to these products:

- IBM Rational Team Concert
- IBM Rational Clear Case
- IBM Software Configuration and Library Management (SCLM)
- IBM ISPW (fast path for Working Set Manager)
- Computer Associates (CA) Endevor Software Change Manager
- CA Librarian
- CA Panvalet
- Serena ChangeMan ZMF
- Microsoft Subversion (an open source product)

Each product integrates with Rational Developer differently.

Rational Team Concert, SCLM, and CA Endevor are supported as shipped. The product features contain built-in views, perspectives, and tooling features to provide integration points between the SCM operational features, such as check-in, check-out, synchronize, build, and so forth, and Rational Developer's source access features. The integration is similar to the integration in 4.3.1, "Remote Systems Explorer: Accessing your z/OS assets" on page 102.

Rational Developer integrates with Microsoft Subversion and Rational Clear Case through a built-in product feature, which is known as *Push to Host*.

Other clients have integrated Rational Developer with their SCM using the Menu Manager TSO Action and the Menu Manager JCL Substitution.

4.6 Integrating with the IBM Problem Determination Tools

The IBM Problem Determination Tools (PD Tools) are powerful and in-depth software offerings that include these offerings and many other products:

- IBM Debug Tool for z/OS
- IBM File Manager for z/OS: QSAM, VSAM, IMS database, and DB2 table editing
- IBM Fault Analyzer for z/OS: ABEND analysis
- IBM Application Performance Analyzer for z/OS

Chapter 6, "IBM Problem Determination Tools" on page 303 describes these tools in-depth.

PD Tools development at IBM is independent of the Rational brand. As of this writing, IBM has not integrated all the PD Tools offerings and functionality into Eclipse and Rational Developer. However, the majority of the developer functionality and practitioner functionality has been integrated and includes Rational Developer integration with the Debug Tool, File Manager, and Fault Analyzer. In this chapter, we describe the integration of these three products with Rational Developer; however, you must educate yourself in-depth to use the tools with your production assets.

4.6.1 Integrating Rational Developer with Debug Tool

IBM Debug Tool is an interactive source-level debugging facility for compiled applications that integrates with Rational Developer within its own Debug Eclipse perspective. Debug Tool and Rational Developer work in a variety of environments (Figure 4-41) and provide testing capabilities for most of the major IBM languages and runtime environments.

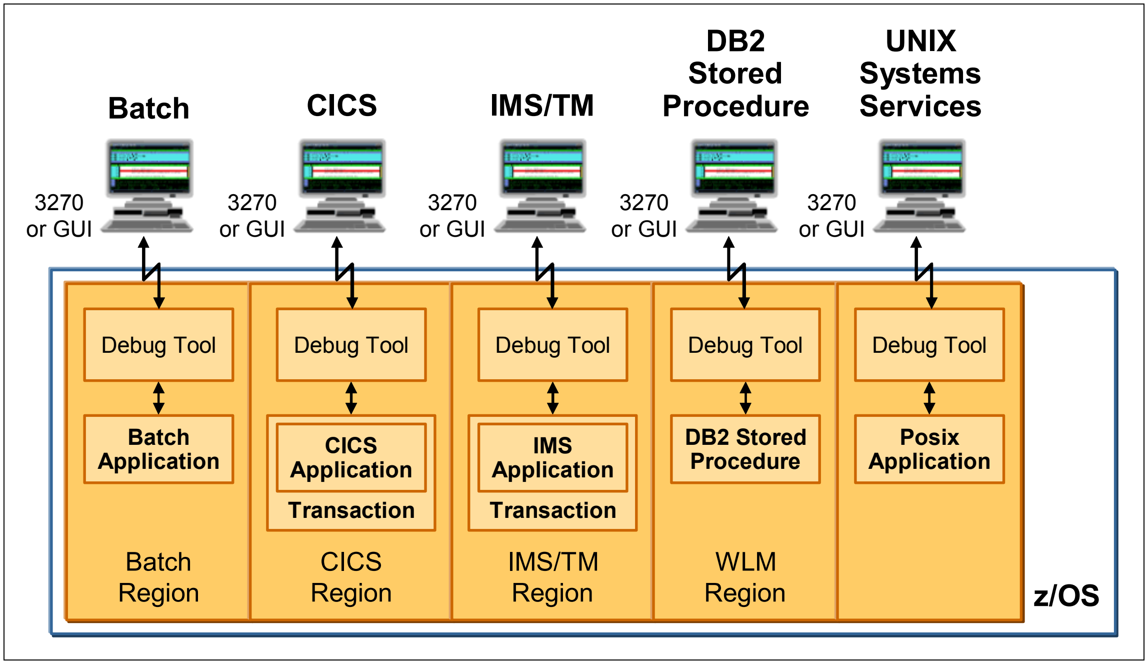


Figure 4-41 Debug Tool supported languages and run times

With Rational Developer and Debug Tool, you can perform these tasks:

- Debug online (CICS or IMS TM) and batch applications
- Debug multiple languages (COBOL, PL/I, HLASM, Java, and so forth), including mixed-language and cross-platform application debugging, for example, where COBOL calls HLASM, or Java calls COBOL CICS
- Display, monitor, and alter program variables
- Set standard types of breakpoints
- View data in hex (EBCDIC) or string values
- Analyze your application dynamically

Analyzing your application dynamically is a key feature. Sometimes, when modifying and attempting to understand the operational mechanics of a program or application, you need to view the code while it executes to understand the control flow or another aspect of the program semantics. For example, static code analysis can limit you, because you only see the transaction codes or the names of the programs that are hard-coded as literals in the code.

However, when analyzing your application dynamically, you can see the transaction codes and the names of the programs in the CALL chain that are referenced within variables. And, often it is easier to understand an application as it executes. You can use Debug Tool to analyze your application dynamically. It allows you to see the application's behavior and to map every aspect of that behavior back to the underlying code.

Debug Tool and Rational Developer integration process

Figure 4-42 depicts the way that the process works. Debug Tool has a listener or daemon that runs on z/OS for debug operations that are launched from your Rational Developer workstation.

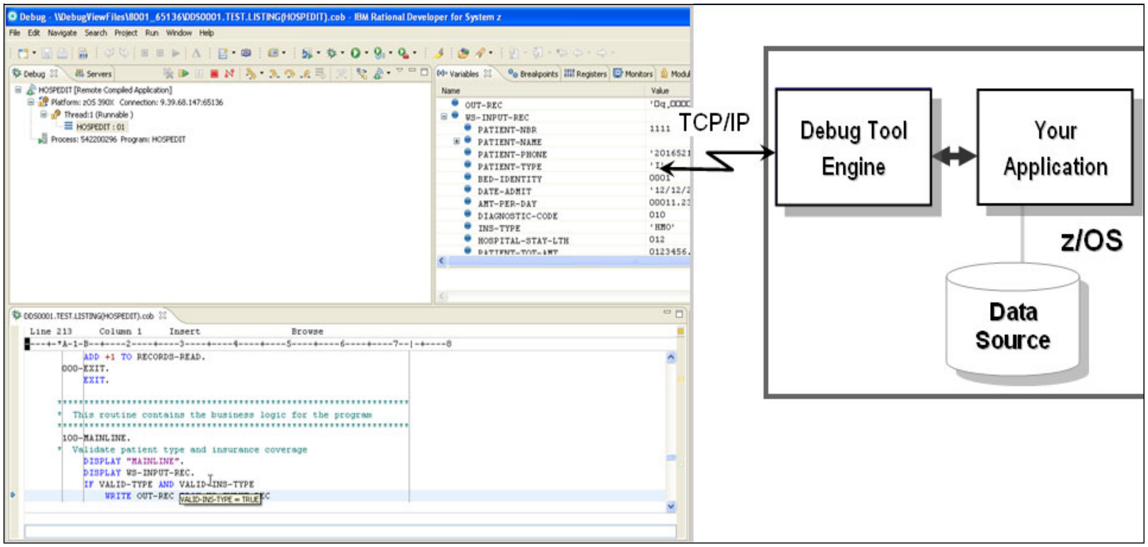


Figure 4-42 Rational Developer and Debug Tool API process: Workstation and z/OS cross-platform debugging environment

This listener, which is shown as the Debug Tool Engine in Figure 4-42, performs these tasks:

- Determines dynamically what action to take on the host based on your Debug session commands:
  - Step
  - Animate (continuous stepping without user intervention)
  - Run to a breakpoint
  - Issue a command
  - Define a breakpoint or variable “watch” monitor
  - Inspect a variable value
- Executes the application statements in the load module on z/OS
- Returns data (the values associated with program variables) in the appropriate Rational Developer Debug Perspective views (Figure 4-43 on page 154) that show this information:
  - Program statements
  - Breakpoints and a few of the dozens of Debug runtime functions:
    - Jumping to a line (unconditional branch)
    - Running to a line (executing statement code)
  - Live, real-time browse access to a VSAM file that is used in the program
  - Real-time browse or edit access to a DB2 table’s rows, from tables referenced in your application code’s EXEC SQL statements; this function allows you to modify table row data dynamically during Debug and to prototype various statement execution paths
  - Monitored expressions
  - Variable values
  - Information about the load module
  - Information about the registers and storage content

Debug Tool and Rational Developer considerations

The following important considerations relate to the Rational Developer and Debug debugging architecture:

- Your code runs from its load module on z/OS. You do not need to offload large, complex files and databases from your existing test platform (off-loading test data is a complicated process).

- Your debugging tools reside in Rational Developer on your workstation, so Rational Developer is the source of all navigation, `find` commands, use of the Outline view, and access to additional views:
  - Figure 4-43 shows many functional views, because the complex nature of production application debugging requires concurrent debug access to database and file I/O results, screen I/O, and semantically rich navigation for understanding the program.
  - Green-screen technology makes concurrent debug strategies difficult or impossible to achieve.
- Compiling for Debug is simple. It is the default in the IBM JCL procedures that are supplied and accessed from Rational Developer.
- Invoking Debug Tool under Rational Developer is easy:
  - For batch applications, you insert your workstation's IP address into a reserved data definition (DD) statement in the JCL step and submit your run JCL. Using the IP address, the Debug Tool Engine invokes debugging within Rational Developer on your workstation.
  - For CICS online applications, you need to set up a profile in a Rational Developer view called DTCN. DTCN describes the information that is required to log in to CICS. You must describe the transactions and load modules that you are about to debug in DTCN.
  - For IMS TM applications, you use TSO to set up a profile (similar to the CICS setup).

Rational Developer's integration with Debug Tool provides a fast and comprehensive, yet simple to set up and invoke, environment for unit and integration test. Many developers have learned that using green-screen debugging technology testing sessions can be onerous to set up. Thus, they often refrain from performing in-depth testing and opt instead to use DISPLAY statements, especially for one-line changes.

The simple and effective approach of Debug and Rational Developer to dynamic code analysis and application debugging encourages developers to not cut corners and to use first-class testing techniques. Debug and Rational Developer not only make you more productive and make you able to accomplish more in less time, but they also improve the quality of your application. With Debug and Rational Developer, you identify more bugs earlier in the testing cycles by simplifying the traceability to the source of the problem.

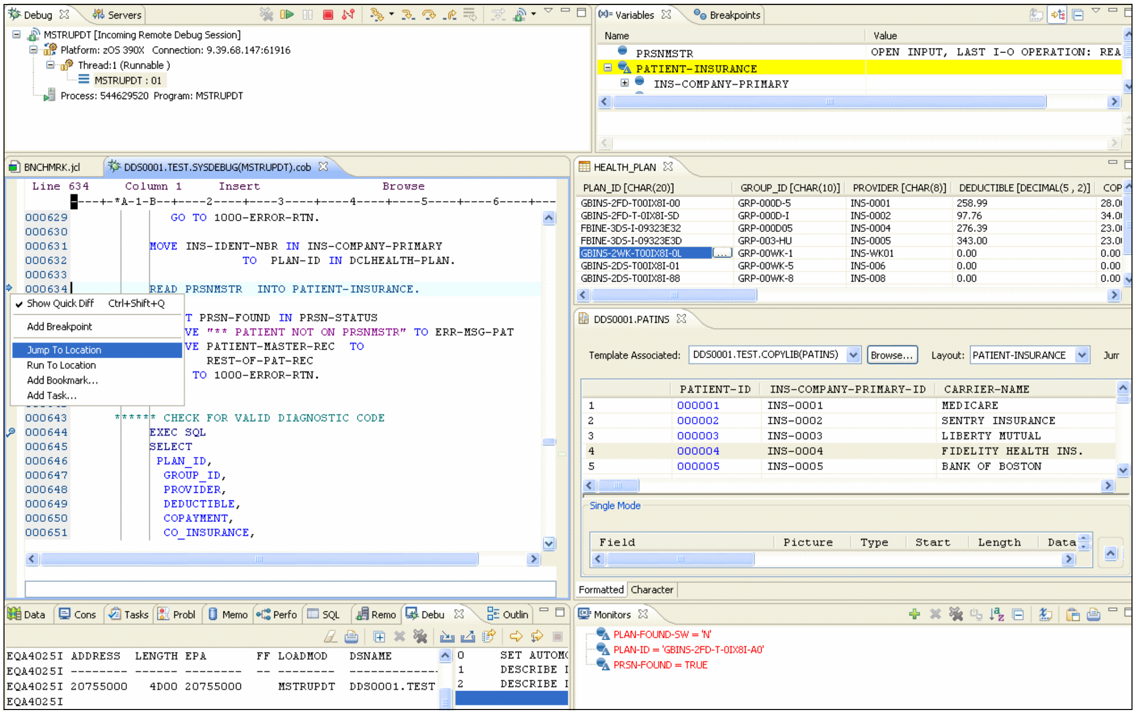


Figure 4-43 Debug Perspective and access to numerous other views related to the program being tested

4.6.2 Rational Developer Integration with File Manager

A common requirement during application development is to browse, search, analyze, and edit QSAM (sequential) and VSAM (indexed, sequential, and relative record) data sets. You have already seen that you use the Data Perspective table editor to work with DB2 test tables and the IMS Universal Driver to work with DL/I databases.

For QSAM and VSAM, you can use the PD Tools File Manager and, specifically, its integration with Rational Developer through Eclipse.

Figure 4-44 on page 156 shows the browse/edit features of File Manager on a VSAM file, and Figure 4-45 on page 157 shows the search/replace functionality. Using these images, note the following functions to view and edit QSAM and VSAM file data:

- You can view file data as a table of records and as an individual record (through a copybook) in Figure 4-45 on page 157.
- You can view file data as a stream of characters under the Character tab in Figure 4-44 on page 156.
- You can open files in full edit or browse-only mode.
- In Edit, you can add and delete records, and you can change field data values.
- You can transfer any number of records down to your workstation and jump around within a file. You control this setting and other settings in the File Manager Preferences page.
- A File Manager History view documents your File Manager data access and allows you to go back to the files to re-access them at any point in time.
- From the Remote Systems Explorer view (not shown), you can use the context menu options, after you have selected a file, for these tasks:
  - Copy data from an existing file with the same characteristics
  - Test data generation
  - Assign a copybook as a template to view a file through logical fields
- For Find/Replace (Figure 4-45 on page 157), you can search within these types of information:
  - Logical fields as defined in the copybook template
  - Excluded records
  - Non-excluded records
  - A file or fields using a regular expression

  You can hide records not found in the file by a search operation.

All of this functionality uses standard Eclipse development skills. You do not need to learn proprietary editing procedures.

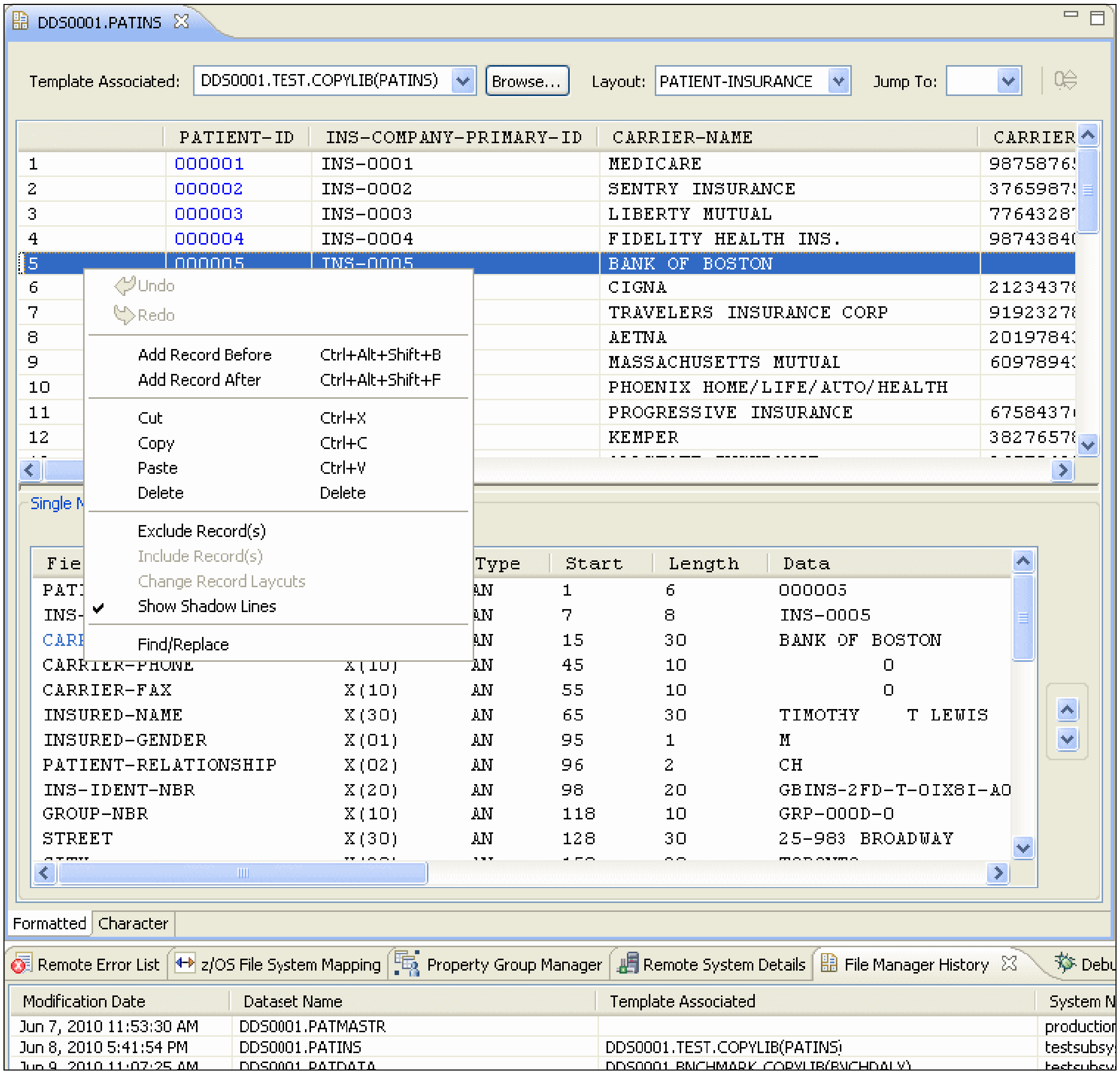


Figure 4-44 File Manager integrated with Rational Developer

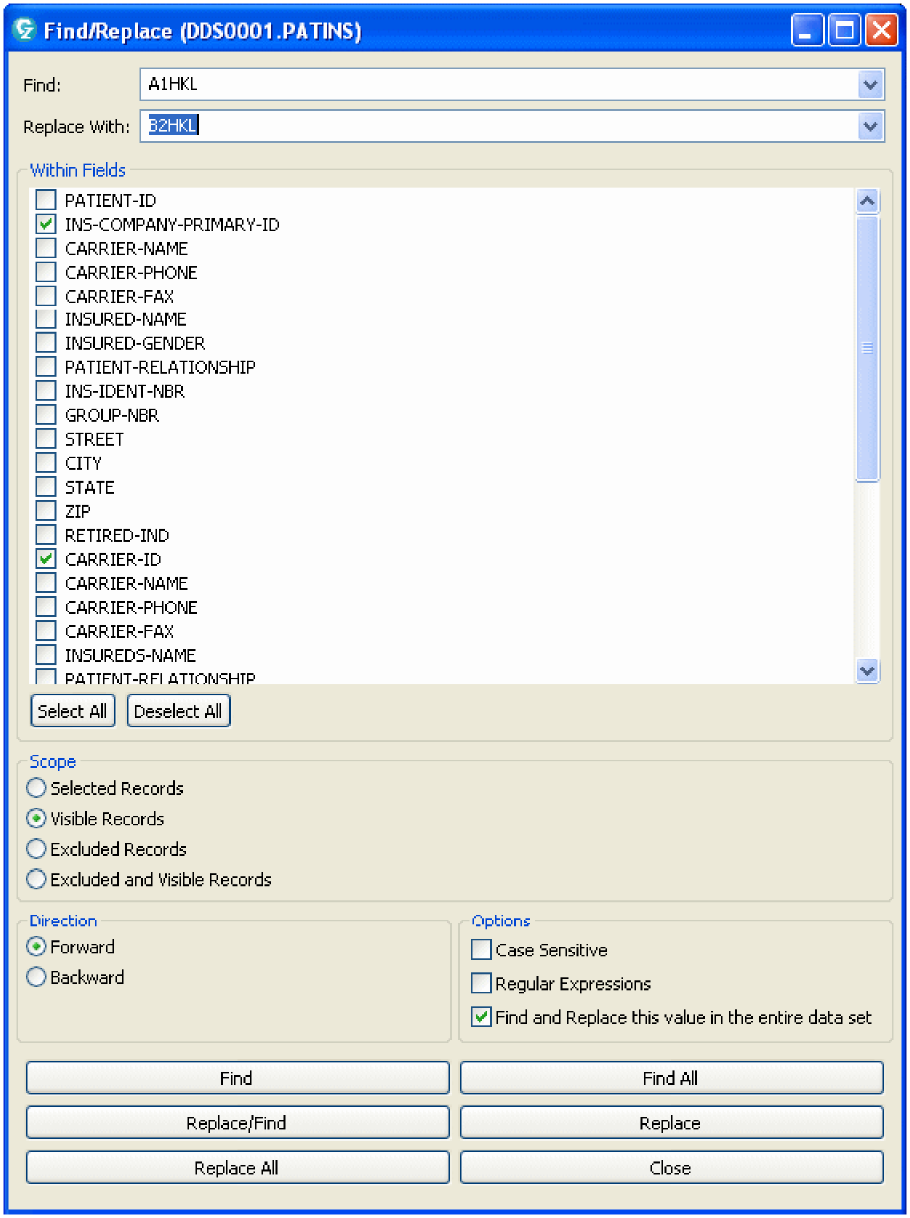


Figure 4-45 Search and replace through logical fields by scope and options

4.6.3 Rational Developer integration with Fault Analyzer

Rational Developer integrates with Fault Analyzer, which is part of the IBM Problem Determination Tool family of products. It runs in both test and production environments with little overhead and offers these functions:

- Provides support for analyzing batch, IMS, and CICS online application and system failures with debugging facilities for all of the IBM mainstream online files and databases:
  - IMS DL/I, DB2, VSAM, IBM Identity Management Services (IDMS), and so forth
  - WebSphere Application Server for z/OS system failures
  - WebSphere MQ application failures
  - Batch (QSAM, VSAM, and DB2) application failures
- Helps you analyze failures when they occur or reanalyze them later
- Expands error messages and codes that apply to your failure with interactive reanalysis and includes a feature for using application-specific messages and codes to supplement those messages and codes that are supplied by IBM
- Creates a fault history file with an interactive display that helps you track and manage application failures
- Starts automatically when an application fails, eliminating the need to recompile programs or change the JCL
- Integrates with Rational Developer to enable developers to diagnose application problems without changing their user interfaces

Figure 4-46 describes how Fault Analyzer operates. When an application ABENDS, a Fault Analyzer listener on z/OS immediately correlates key information about the ABEND into a Fault Analysis report (Figure 4-47). Fault Analyzer also creates a history file with critical Fault Analysis information and DUMP data, including registers. Developers use this history file when solving ABENDs. Managers use this history file to track the health of an application (traceability).

The Fault Analyzer report describes the following types of information:

- The nature of the ABEND
- An ABEND lookup document is provided that describes how and, more importantly, why common runtime ABENDs occur and gives a little assistance in identifying the root cause
- The load module, job, or transaction that was running when the ABEND occurred
- The program in which the ABEND occurred
- The statement that forced the ABEND
- Any variables and variable values that are considered evidence or points of interest in understanding the nature of the ABEND
- Hyperlinks back to the program source for ease of use

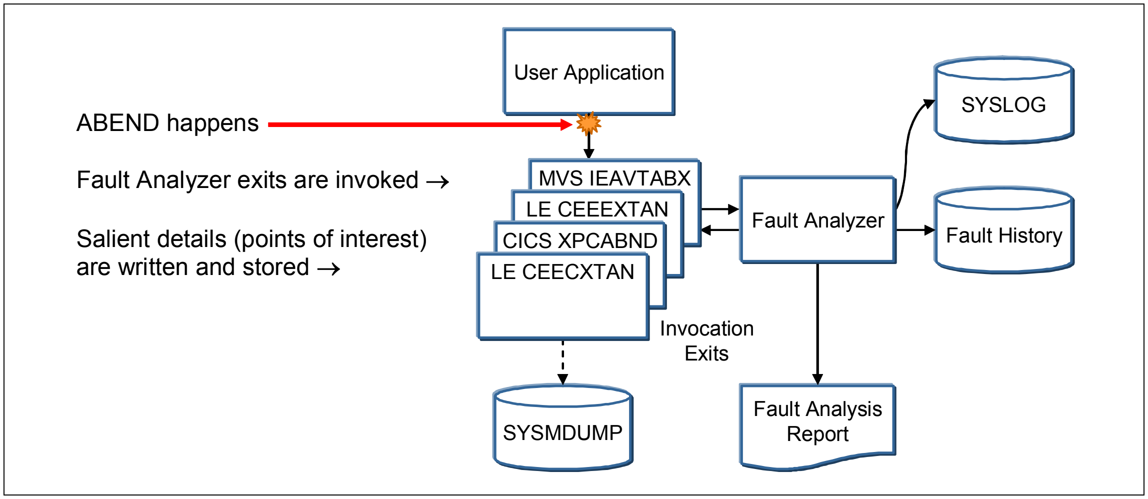


Figure 4-46 Fault Analyzer process at run time showing events and Fault Analyzer artifacts

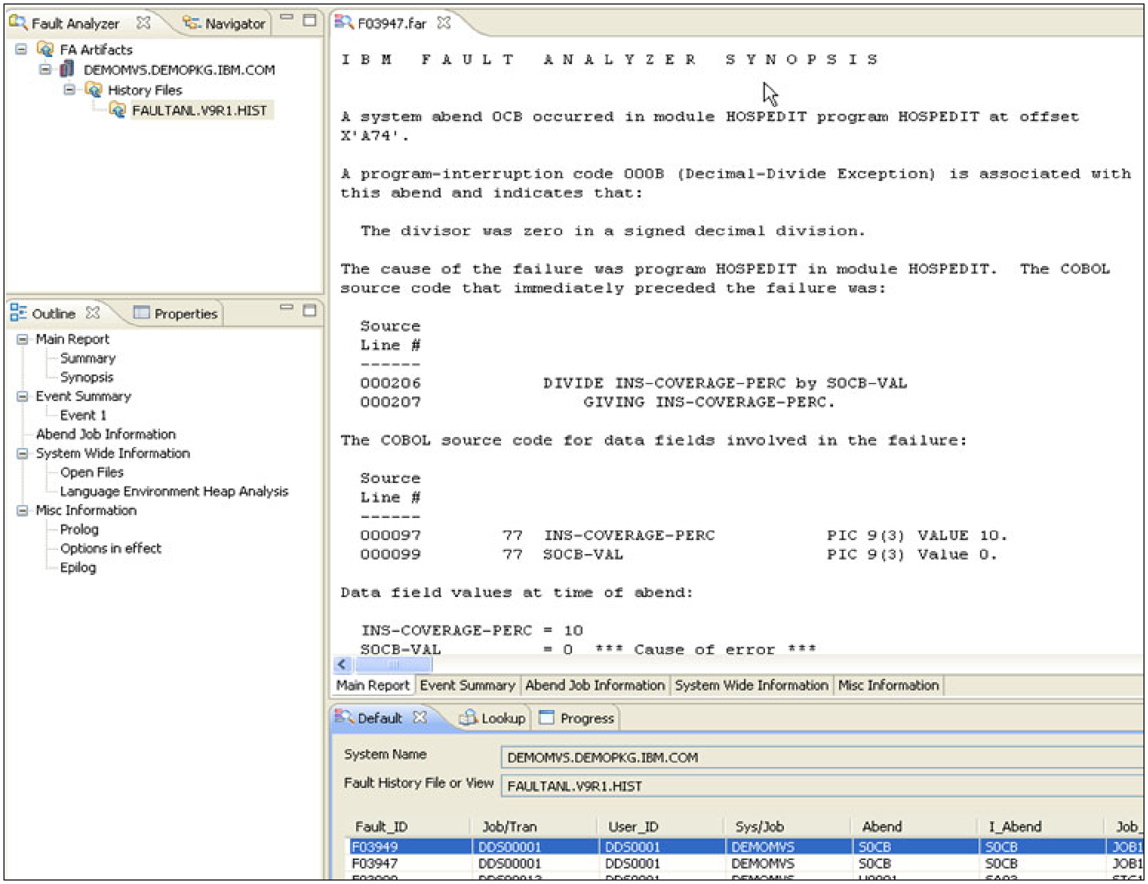


Figure 4-47 Fault Analyzer perspective and report showing why (and where) an ABEND occurred inside a program and formatting the necessary information to respond to the ABEND situation

Fault Analyzer allows the less-experienced members of your staff to work productively with application ABENDs.

4.7 Integration with Rational Asset Analyzer

As part of the research for this book with IBM clients worldwide, we asked z/OS application developers:

> "*How does your professional day break out among analyzing and analysis-related activities, coding/compiling, and testing and testing-related activities, such as managing test data, as a percentage*?"

We collected the following results as an industry average:

- 46% - Analyzing and analysis-related activities
- 26% - Coding/compiling
- 28% - Testing and testing-related activities

So, by almost a 2 to 1 margin, analysis consumed the largest part of developer time in our survey. This result is not surprising in the z/OS application world, because most z/OS “development” consists of maintenance, application support, and making modifications to an existing codebase. This type of activity requires much more understanding than typing. One maintenance programmer summed up this environment well, “*I just spent three days making a five-line change.*”

There are many aspects of software analysis. Most types of software analysis are out of the scope for this book; however, we describe dynamic application analysis and static application analysis.

4.7.1 Dynamic application analysis

Debug Tool, which we describe in 4.6.1, “Integrating Rational Developer with Debug Tool” on page 150, allows you to view the behavior of your batch or online application while it executes under z/OS and to map that behavior back to the underlying code. We refer to this tracing of an application’s control flow to its program source and learning the code by watching as it executes as *dynamic application analysis*. The analysis occurs in real time while the application executes dynamically on the machine. We explain more information about the PD Tools in Chapter 6, “IBM Problem Determination Tools” on page 303 of this book and in the IBM Redbooks publication, *IBM Problem Determination Tools for z/OS*, SG24-7918-00.

Additionally, you can use CICS Interdependency Analyzer in dynamic application analysis. See the IBM Redbooks publication, *IBM CICS Interdependency Analyzer, SG24-6458-01*, for more details on CICS Interdependency Analyzer.

4.7.2 Static application analysis

You perform *static application analysis* from the static source code. Traditionally without Rational Asset Analyzer, this type of analysis was a time-consuming, error-prone, and onerous process of reading listing files, going from sorted cross-reference (SX-REF) line numbers back to the source. Or, you used ISPF 3.4 or 3.14 to perform multiple FIND commands.

Both of these approaches are expensive and unpleasant, which is why you want to use Rational Asset Analyzer.

4.7.3 Rational Asset Analyzer

Rational Asset Analyzer is the IBM Rational software group's static application analysis package for understanding programs, application discovery, code comprehension, and visual understanding. Rational Asset Analyzer simplifies the task of assimilating the complex and enormous processing semantics within a production-quality module. Rational Asset Analyzer helps you follow the trail of a data value as it proceeds from variable to variable to file to variable to database across an application. Rational Asset Analyzer is especially helpful when compared to the same time and effort that it takes to accomplish the same goal using ISPF and source listings.

Rational Asset Analyzer is server-based and capable of materializing an enterprise-wide, cross-platform, and cross-application view of your software's processing semantics. Often, software and project architects, systems and business analysts, and project lead technical individuals monopolize Rational Asset Analyzer for their enterprise, division, department, and project scoping and analysis. IBM offers Rational Asset Analyzer integration to the COBOL, PL/I, and Assembler programmers for analysis work during their development.

Anyone who has programmed at the production level knows that the analysis does not end when the project life cycle that is known as analysis ends. For example, if you modify a complex calculation and while you modify the COBOL COMPUTE statement, you realize that you do not know where a specific value came from, starting with the database read and ending at the COMPUTE statement. Then, you must analyze the code, because you know from experience that if you do not understand the program logic, there will be problems. You must analyze until you understand the code.

Similarly, you might debug a unit test scenario, and the program logic executes an IF statement that does not behave in the way that you expected. You do not continue testing. You return to analysis and dig deeper into the code until you have an accurate understanding of the control flow and conditions.

Analysis is pervasive throughout the development life cycle; therefore, you need the best analysis tools you can get, such as Rational Asset Analyzer integration.

4.7.4 Rational Asset Analyzer integration

Since Rational Developer V7.5, Rational Asset Analyzer has offered a Rational Developer and Eclipse plug-in with a subset of Rational Asset Analyzer functionality that is called *Rational Asset Analyzer integration*. This functionality is no charge if you own Rational Asset Analyzer and Rational Developer.

Rational Asset Analyzer integration consists of the ability to designate a Rational Developer local z/OS project as enabled for Rational Asset Analyzer integration support. This support provides four major functional options:

- Integration with Rational Developer. Rational Asset Analyzer integration provides an analysis engine. Rational Developer provides the development IDE. Their integration provides an optimal workbench for z/OS maintenance and support activities.
- Rational Asset Analyzer Program Tree View generation, which is used in program-level analysis.
- The ability to detect *unreachable or dead code*. Unreachable code is code that can never be executed inside your program due to the current control flow. Figure 4-48 on page 167 shows a diagram of how Rational Asset Analyzer integration depicts unreachable code. You follow these steps:
  a. Select the program that you want to analyze.
  b. Right-click the program, and select **Rational Asset Analyzer Integration** → **Show unreachable code**.
  c. Performing this action opens the Rational Developer LPEX editor and allows you to perform the following tasks, among others:
    - Modify the source with Rational Developer.
    - Analyze the program further by using the Rational Asset Analyzer integration control flow diagram (Figure 4-48 on page 167).
    - Document (add Rational Developer bookmarks or tasks).
- The ability to rescan changes dynamically. You can update your programs, JCL, DBD/PSB, and so forth dynamically, and, after rescanning, the Rational Asset Analyzer models reflect your changes. To rescan dynamically, the developer’s workstation must have a copy of Rational Asset Analyzer. Otherwise, possible source and version control management issues occur if multiple individuals update a centralized server-based model.

4.7.5 The program tree view

The program tree view provides COBOL programmers with both broad and in-depth analysis tooling for COBOL programs beyond the functionality in Rational Developer. From Figure 4-48 on page 167, you can see several features of the program tree view, including these features:

- The program diagram (see Figure 4-49 on page 168) is a snapshot of all external program elements, such as called programs, files, and database access, including the direction of the I/O. You can see input I/O versus output I/O as designated by the connection arrows.
- The control flow diagram (see Figure 4-48 on page 167) is a hierarchical chart of the program's PROCEDURE DIVISION that is hyperlinked so that a developer can click a paragraph or section, and the editor synchronizes the code with the hierarchical chart.

  Most programmers learn a program in two directions: Top-down and bottom-up. The control flow diagram is the top-down view. Without this technology, it is difficult and slow to create this view. The bottom-up view describes the details of each paragraph. By opening both views simultaneously, you can learn the program logic easily, quickly, and with fewer passes.

  You can combine the control flow diagram with the Rational Developer perform hierarchy for an additional, more linear view of your source logic flow.
- The data element table (see Figure 4-50 on page 169) is a sorted cross-reference of all the variables inside a program, and the variables are hyperlinked to the program source.

  Additionally, the data element table provides "where used" analysis and "how used" analysis to allow you to isolate and view, for example, only the statements that modify a certain variable, reference a certain variable, or both.

  The data element table is extremely useful functionality for maintenance programmers who spend an inordinate amount of time performing detailed analyses of an ABEND, such as an 0C7 or 0C4, due to invalid data values in a field. To track this type of a problem, programmers must look up all variable references and indirect references (READ INTO, SELECT, IMS Get, and so forth at the record or group level), and they must ascertain the specific instructions that modify a field's contents out of the potentially thousands of references in a large program.

- Files needed to build, which is a list of all necessary copybooks to compile the program successfully.
- Top level data items, which is all 01 records in the program.
- All external program CALLs to other programs.
- The run-unit, which is a diagram of the transaction or batch job in which the program participates.
- DB2 tables and table columns, including the details of the tables, views, and columns that the program accesses (also IMS databases, where applicable).
- Data sets and data stores, including details about the QSAM and VSAM files that the program accesses and the file directories (FDs) to which they are assigned internally.
- The Rational Asset Analyzer Details is an immediate link to the Rational Asset Analyzer product with dozens of other useful analysis functions, such as the JCL batch job diagram (Figure 4-51 on page 170).

Important: All the reports and diagrams are Scalable Vector Graphics (SVG) files that can be saved or added to documentation. When you save an SVG graphic, you actually save the links that comprise the file, which means that you can use the saved graphic to hyperlink back to the Rational Asset Analyzer and Rational Asset Analyzer integration reports (assuming that Rational Developer is running on your workstation).

4.7.6 Working with Rational Asset Analyzer integration

When you first work with the Enterprise Modernization (EM) Workbench, it has so much functionality that it can be daunting to know where to start, what to use, when to use Rational Developer versus Rational Asset Analyzer integration, and so forth. We break it down into five standard Rational Asset Analyzer integration use case functions:

- Facilitating top-down program comprehension
- Control flow analysis
- Data flow analysis
- Impact analysis
- Documenting the application with Rational Asset Analyzer integration

Facilitating top-down program understanding

Starting with the program source that is loaded into Rational Developer, you right-click the program source and select **Rational Asset Analyzer integration** → **Program tree view**. From the Program tree view, you right-click and select **Show program diagram** (Figure 4-49 on page 168).

Here, you can learn about an individual program from a high level and get an understanding of program I/O and calls to subroutines. If you are studying a CICS or IMS TM transaction, use the Run unit[1] diagram (not shown). If you are studying a z/OS batch job, you can use the Batch job diagram (Figure 4-51 on page 170) to gain a clear understanding of these components:

- Files
- I/O activity
- Job step dependencies
- Calls to subroutines and other run units

Control flow analysis

Going from a high-level view to a low-level view, you can use the control flow diagram (Figure 4-48 on page 167) to understand how each program is hierarchically structured from all the PERFORM chains and other branching statements (possibly, GO TOs).

Follow these steps to analyze the program further:

1. Typically, you click a starting paragraph in the control flow diagram, which synchronizes the source in the Rational Developer editor.
2. You read the individual paragraph source in the editor to gain a bottom-up understanding of the content in each paragraph, reading statement by statement when necessary, but still having the program's structure to refer to, which shows in the control flow diagram.
3. You might combine this approach with the Rational Developer perform hierarchy at any point in the process to detour to see a subset of the paragraphs.

[1] A run unit is the operational (load module) level view starting from each program or transaction and ending at GOBACK for that individual program.

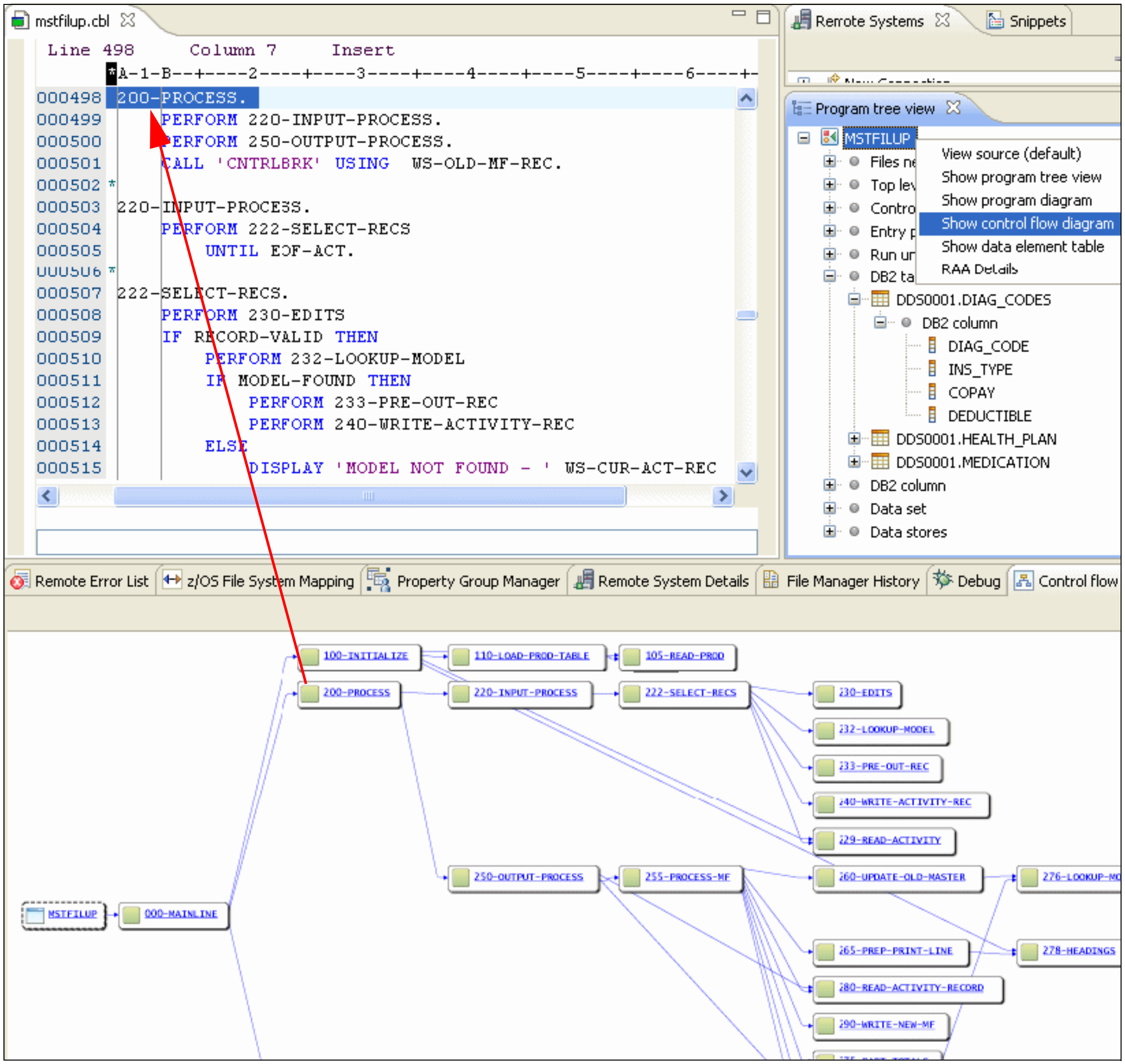


Figure 4-48   Rational Asset Analyzer integration's program tree view showing the control flow diagram synchronized to the program source file

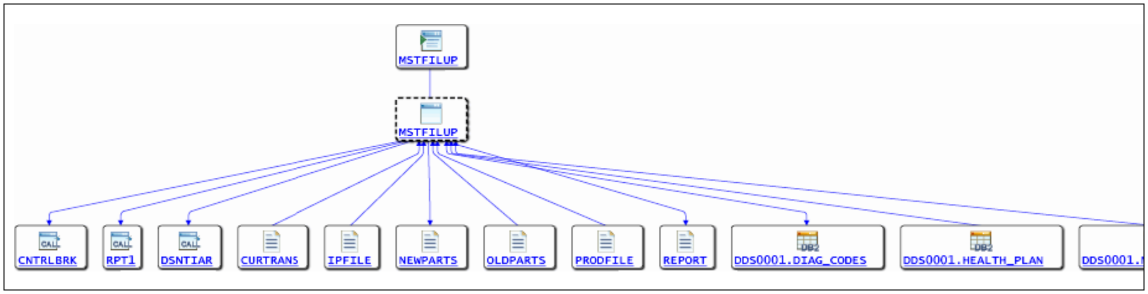


Figure 4-49 Rational Asset Analyzer integration program diagram with arrows showing the I/O access direction of various files and databases

Data flow analysis

A common requirement for maintenance and support projects is to track the progression of variable values as they are processed, moved, or made part of arithmetic operations, a file, a database, MQ, or screen I/O. We refer to this tracking as *data flow analysis*.

Performing data flow analysis by using ISPF searches, source listing lookups, and compiler cross-references is lengthy and error-prone. To simplify this work by using Rational Asset Analyzer integration, you can perform these tasks:

1. Starting from the original or “seed” variable or literal, you select the value from the data element table, right-click, and select **References and Modifications** (Figure 4-50).
2. This step yields the Search results view, as shown in Figure 4-50.
3. Because you need to save this list and refer back to it during your work, you “pin” the results view, which persists the list of hyperlinked statements.
4. You read each statement by clicking the down arrow (Figure 4-50) under the Program tree view tab in the lower right of the screen capture to determine if that statement referred to additional variables requiring additional searches.
5. If additional searches are required, you begin the process by searching on a new variable in the data element table, pinning results, clicking through the list, reading, and so forth.

In this approach, you follow a tree that is composed of branches as variables and read each branch to see if there are offshoots (additional branches).

Rational Asset Analyzer integration has additional project-wide search capabilities that can be used to extend your search analysis beyond one program. Figure 4-56 on page 176 shows these capabilities, where you can see the Search window, that allow you to search for statements, in this case, that modify the variable patient-copay across all the source files in a local or remote project.

The ability to filter hundreds of thousands of lines of source code to pinpoint your exact searched-for request can reduce the time that it takes to perform this type of complex task dramatically in large systems.

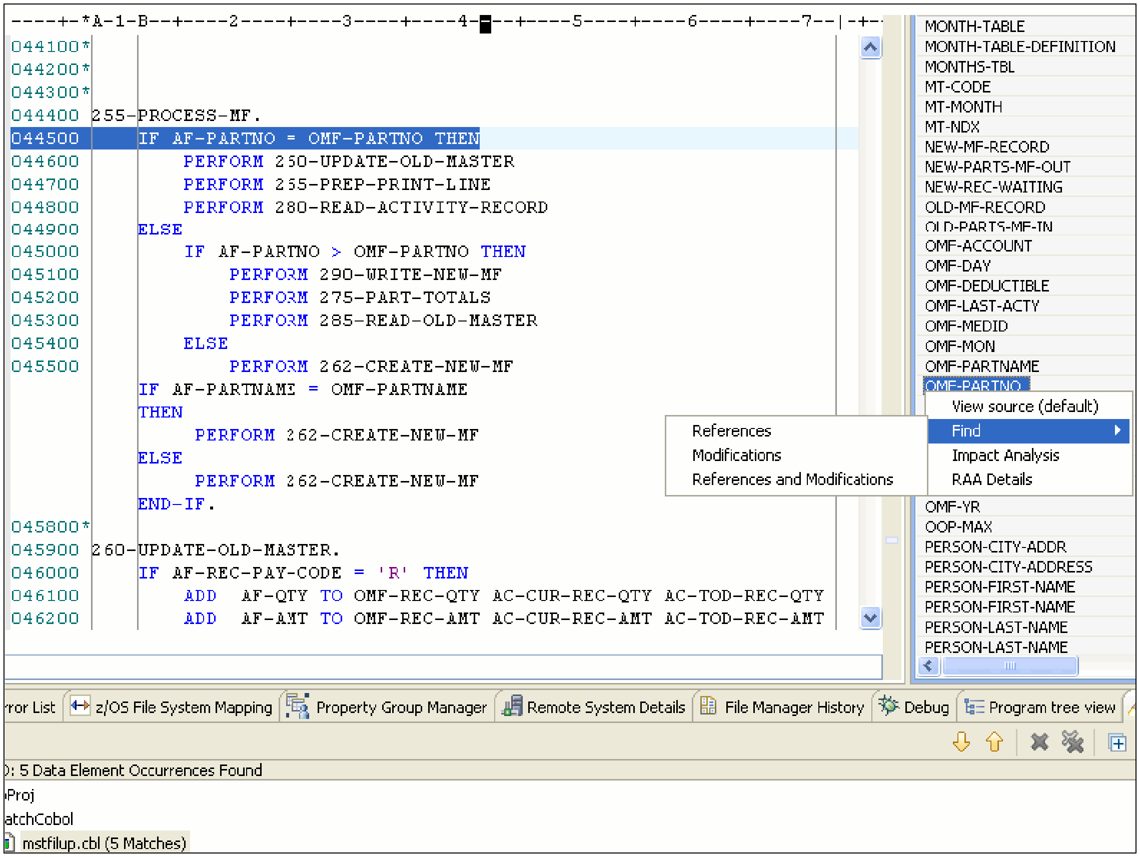


Figure 4-50 The data element table that is used for semantically rich search and analysis

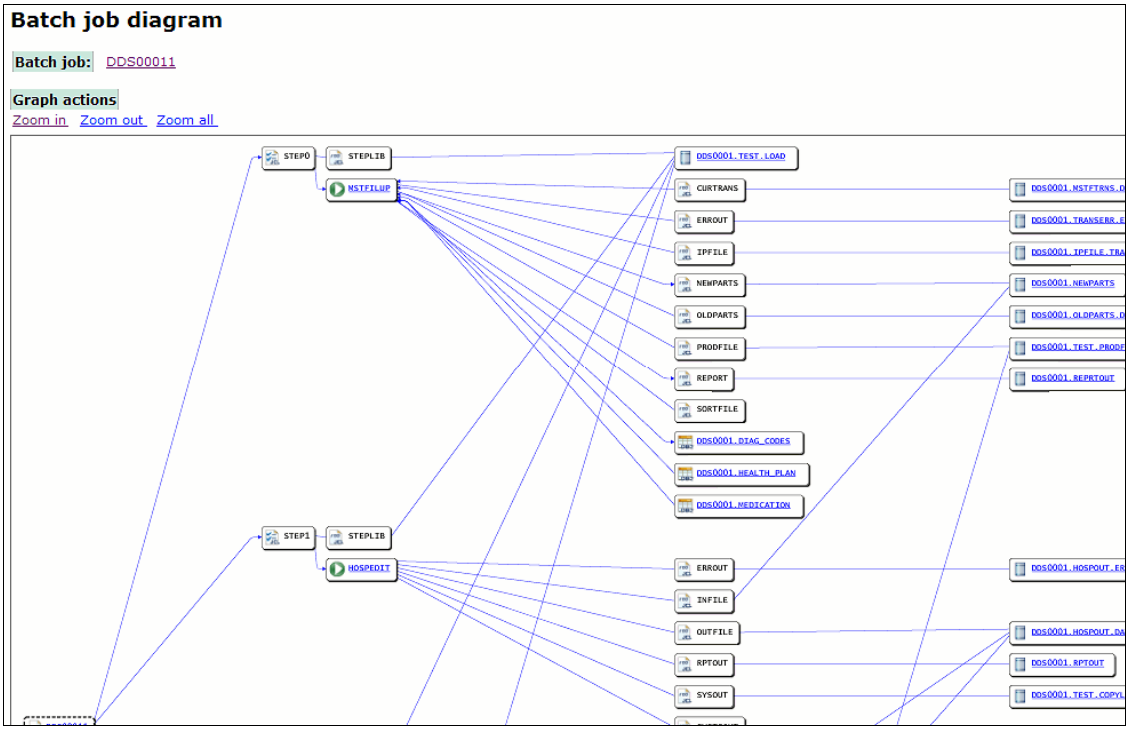


Figure 4-51 Rational Asset Analyzer Batch job diagram that is accessed from Rational Asset Analyzer integration through Rational Asset Analyzer Details

Impact analysis

Another approach for solving the problem of data flow analysis is to use impact analysis. Rational Asset Analyzer integration's *impact analysis* feature builds the tree structure as a report. The Impact Analysis report (Figure 4-52) shows all program elements, actually all application-scoped elements, that might be affected by a modification or change to a file, program statement, range of program statements, or a DB2 table or column. This type of analysis task is among the most in-depth and challenging types of software maintenance activities. Automating impact analysis by using static analysis tooling is automatically beneficial and highly valuable.

Follow these steps to generate an Impact Analysis report:

1. Start by selecting a seed variable.
2. Right-click the variable from the Rational Asset Analyzer integration:
   - Program tree view
   - Data element table (Figure 4-52)
   - Any element in the IBM RAA® Details view
3. Click **Impact Analysis**.

From Figure 4-52 on page 172, you can see that a report is produced as an Eclipse view. Each element in this report hyperlinks back to one of these points:

- A source code line in a program within the scope of the analysis (see the impacted data elements in Figure 4-52 on page 172)
- An element that you can analyze further with either Rational Asset Analyzer integration or Rational Asset Analyzer using the Rational Asset Analyzer Details (RAA Details) option:
  - Figure 4-53 on page 173, which shows a graphical representation of the possible ramifications of changing the variable from a high-level view
  - Figure 4-54 on page 174, which shows the details and an in-depth view of the work needed to analyze the change completely

> **Hyperlinks:** You can copy the hyperlinked elements to any standard documentation, and the elements remain hyperlinked to the Rational Asset Analyzer model even if you have copied them into a spreadsheet.

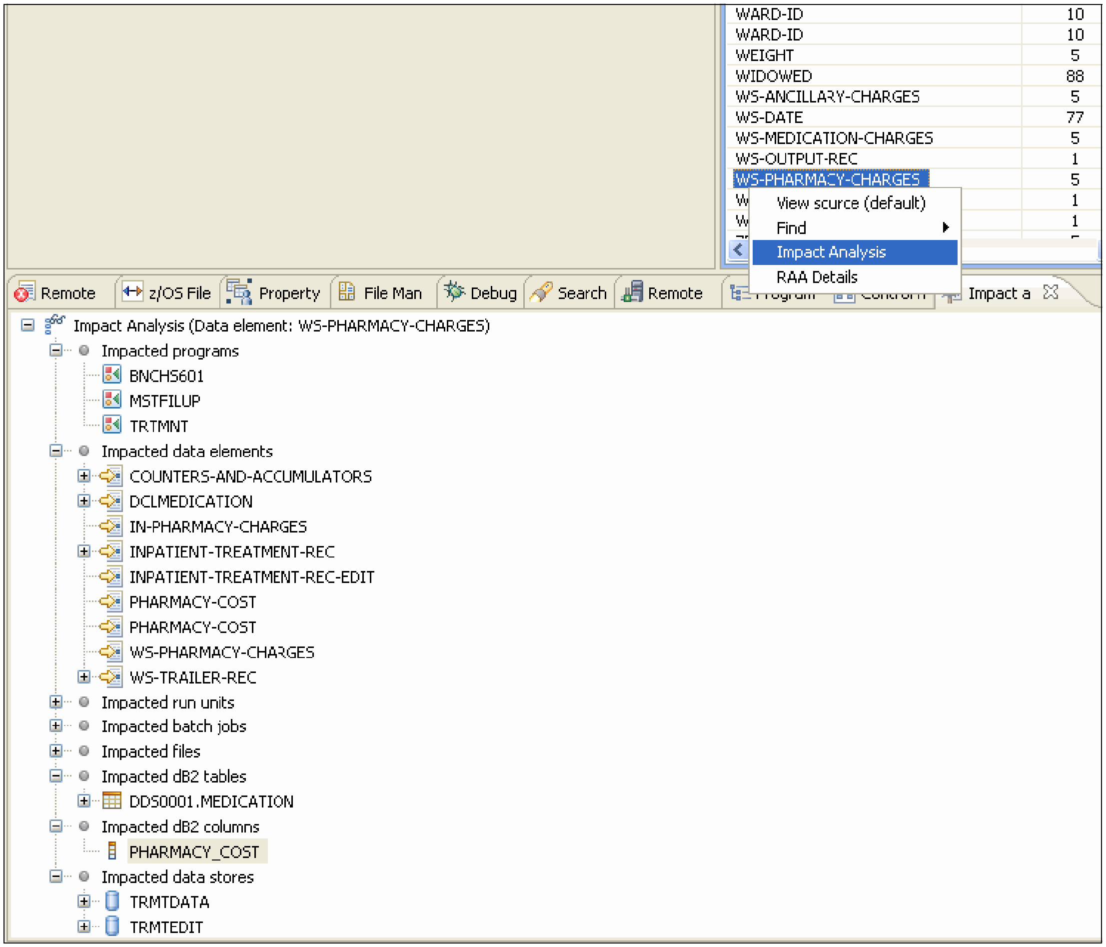


Figure 4-52   Rational Asset Analyzer integration Impact Analysis report

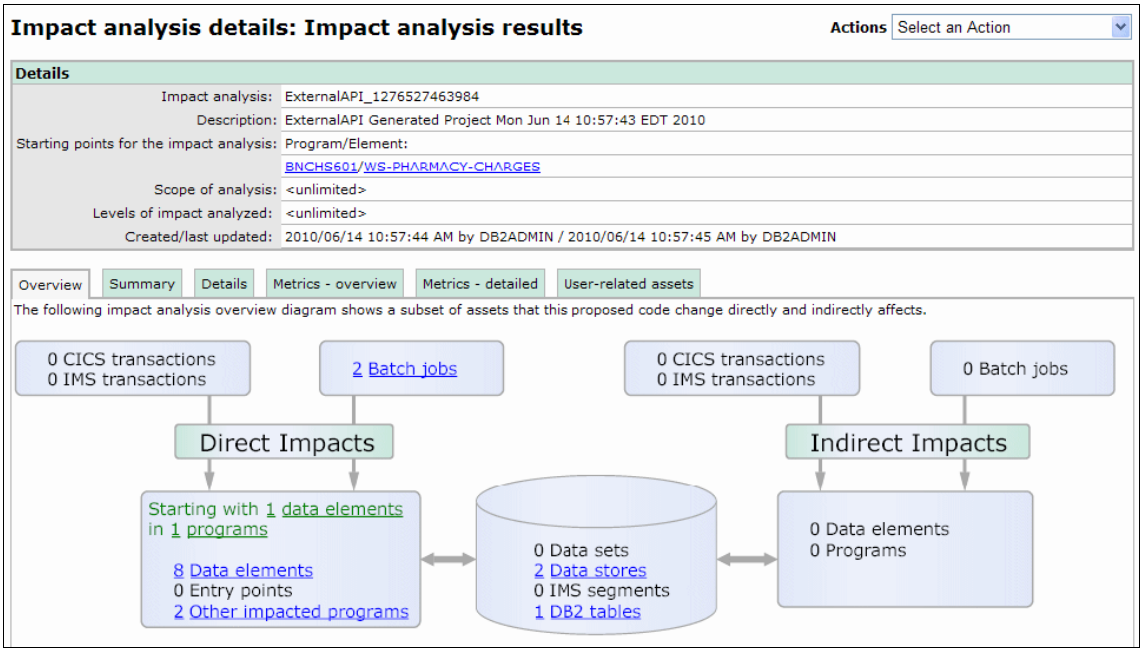
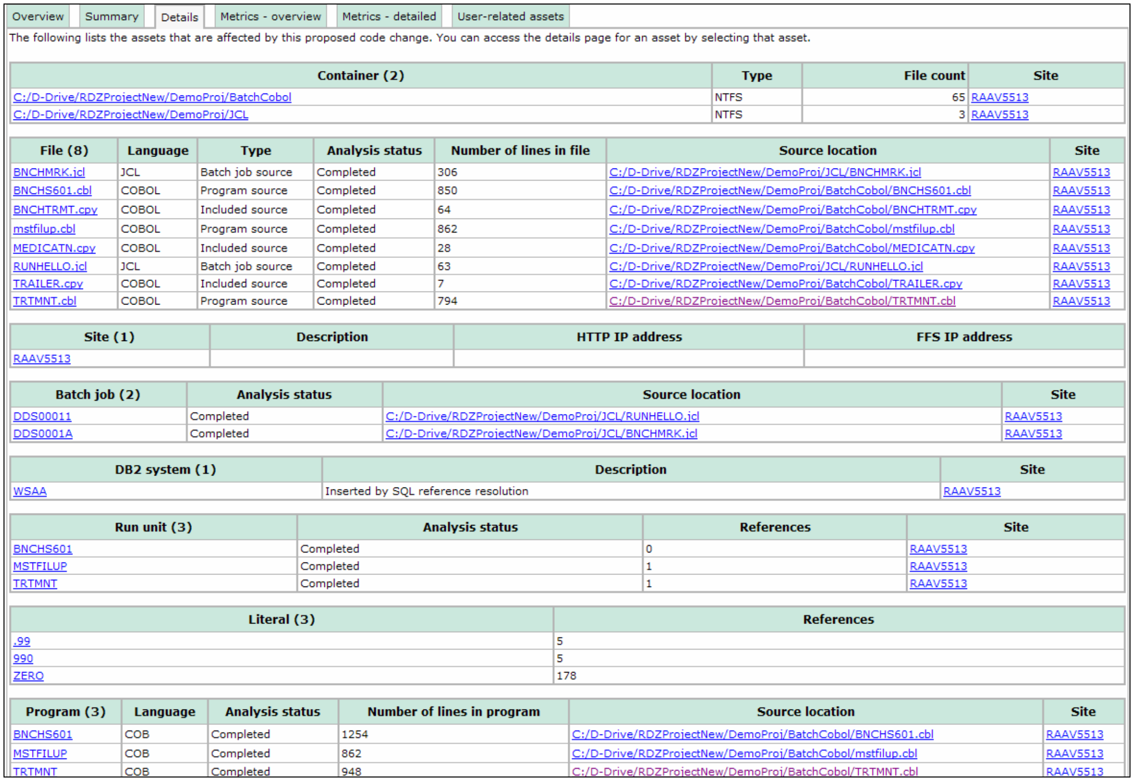


Figure 4-53 Rational Asset Analyzer integration impact analysis results under the RAA Details option



Overview | Summary | Details | Metrics - overview | Metrics - detailed | User-related assets

The following lists the assets that are affected by this proposed code change. You can access the details page for an asset by selecting that asset.

Container (2)	Type	File count	Site
C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol	NTFS	65	RAAV5513
C:/D-Drive/RDZProjectNew/DemoProj/JCL	NTFS	3	RAAV5513

File (8)	Language	Type	Analysis status	Number of lines in file	Source location	Site
BNCHMRK.jcl	JCL	Batch job source	Completed	306	C:/D-Drive/RDZProjectNew/DemoProj/JCL/BNCHMRK.jcl	RAAV5513
BNCHS601.cbl	COBOL	Program source	Completed	850	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/BNCHS601.cbl	RAAV5513
BNCHTRMT.cpy	COBOL	Included source	Completed	64	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/BNCHTRMT.cpy	RAAV5513
mstfilup.cbl	COBOL	Program source	Completed	862	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/mstfilup.cbl	RAAV5513
MEDICATN.cpy	COBOL	Included source	Completed	28	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/MEDICATN.cpy	RAAV5513
RUNHELLO.jcl	JCL	Batch job source	Completed	63	C:/D-Drive/RDZProjectNew/DemoProj/JCL/RUNHELLO.jcl	RAAV5513
TRAILER.cpy	COBOL	Included source	Completed	7	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/TRAILER.cpy	RAAV5513
TRTMNT.cbl	COBOL	Program source	Completed	794	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/TRTMNT.cbl	RAAV5513

Site (1)	Description	HTTP IP address	FFS IP address
RAAV5513			

Batch job (2)	Analysis status	Source location	Site
DDS00011	Completed	C:/D-Drive/RDZProjectNew/DemoProj/JCL/RUNHELLO.jcl	RAAV5513
DDS0001A	Completed	C:/D-Drive/RDZProjectNew/DemoProj/JCL/BNCHMRK.jcl	RAAV5513

DB2 system (1)	Description	Site
WSAA	Inserted by SQL reference resolution	RAAV5513

Run unit (3)	Analysis status	References	Site
BNCHS601	Completed	0	RAAV5513
MSTFILUP	Completed	1	RAAV5513
TRTMNT	Completed	1	RAAV5513

Literal (3)	References
.99	5
990	5
ZERO	178

Program (3)	Language	Analysis status	Number of lines in program	Source location	Site
BNCHS601	COB	Completed	1254	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/BNCHS601.cbl	RAAV5513
MSTFILUP	COB	Completed	862	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/mstfilup.cbl	RAAV5513
TRTMNT	COB	Completed	948	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/TRTMNT.cbl	RAAV5513

Figure 4-54 Rational Asset Analyzer integration impact analysis Details tab (partial results and hyperlinks to source files)

Documenting your application using Rational Asset Analyzer integration

In addition to these analysis usage models, developers commonly include several of the Rational Asset Analyzer integration diagrams and reports with their system documentation. All the diagrams are SVG files, and they retain their hyperlinks back to the source elements from which they were derived.

You can incorporate Rational Asset Analyzer integration as an extremely simple and effective means of collaborating and sharing analysis workflow. You can copy and paste the Impact Analysis detail report (Figure 4-54) to any rich-text document processor, such as Microsoft Word, Microsoft Excel, or IBM Lotus® Symphony™, to provide project teams with an ideal working document for maintenance tasks.

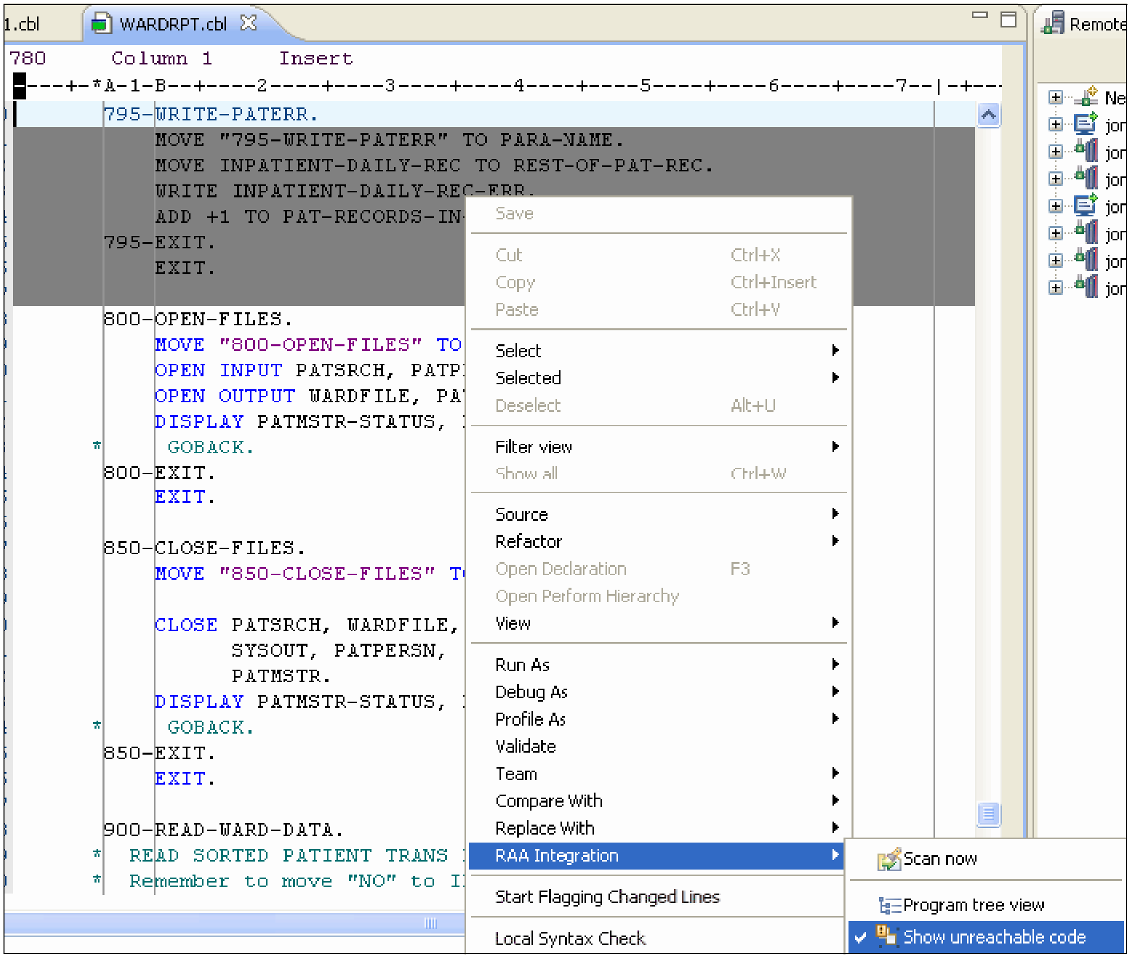


Figure 4-55 Rational Asset Analyzer unreachable code and Rational Developer editor integration

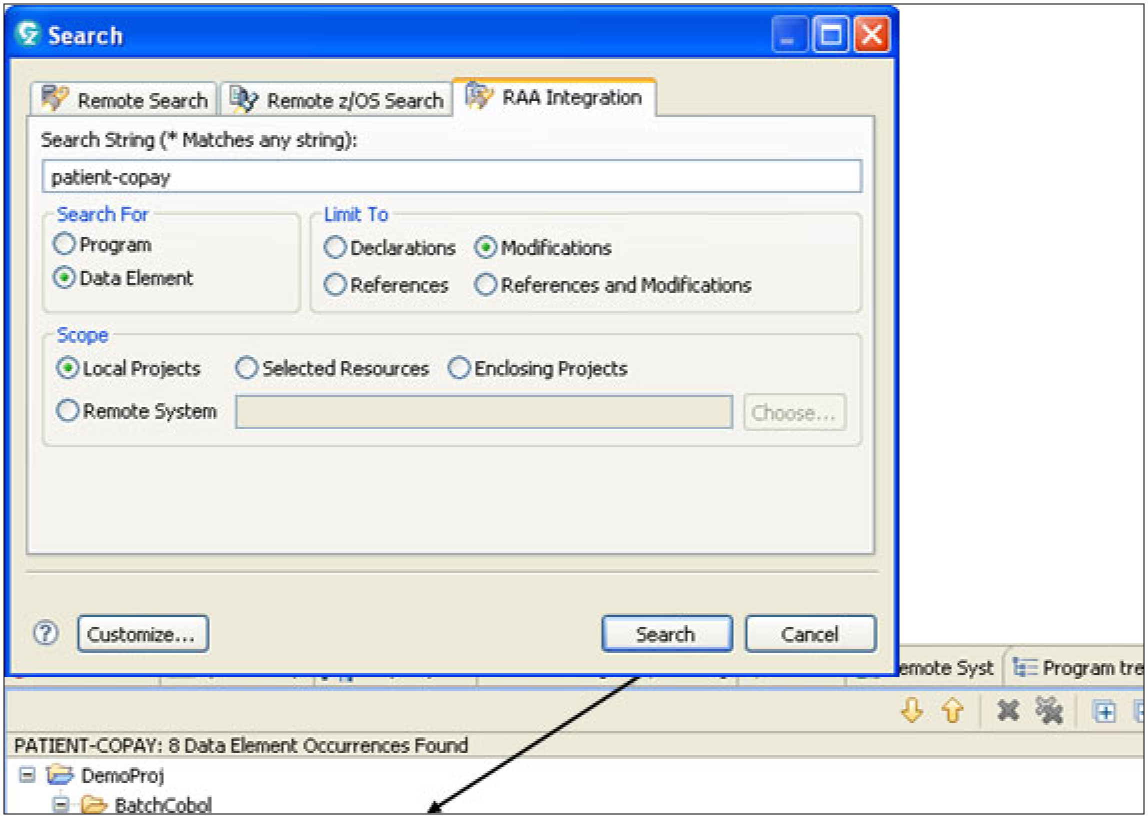


Figure 4-56 Rational Asset Analyzer integration Search for statements that modify the contents of any variable named patient-copay

4.8 Host integration with Menu Manager

In this section, we describe the following topics:

- Menu Manager in the Rational Developer context
- Menu Manager functions
- Maintaining files, menus, and actions
- Creating a simple remote file action in a step-by-step guide
- Creating simple local file action in a step-by-step guide

4.8.1 Menu Manager in the Rational Developer context

Menu Manager is a Rational Developer for System z mechanism for integrating traditional and modern actions into the Eclipse IDE to preserve past developments and take advantage of new technologies.

Over time, TSO and ISPF developers have developed their own set of tools, utilities, commands, and macros. This set, collectively named *Actions*, is part of the practitioner's toolbox when moving to work with Rational Developer. Among those actions, developers frequently use the following items:

- TSO/REXX execs
- Batch jobs
- Emulator's macros and commands
- Workstation utilities and commands
- Web shortcuts

Using Menu Manager, developers can create simple or complex actions to provide the same functionality in their Rational Developer for System z workspace. Developers define and then add these actions to the right-click context menu as menu items, providing easy-to-use tools that can be shared among developers.

4.8.2 Menu Manager functions

Menu Manager provides a graphical interface through the workbench Preferences page to create and maintain customized menus for a project, subproject, or a file. Using this interface, a user can perform these tasks:

- Associate menus with a specific project, subproject, or file
- Include or exclude menu items from a menu
- Override existing actions within menus
- Create simple local actions, for example:
  - Invoke a local utility, such as Notepad or Microsoft Excel, against the currently active file
  - Run a Java class to manipulate data that is currently in the editor
- Create simple remote actions, for example:
  - Submit a job with dynamic variable substitution
  - Execute a REXX exec against the currently active data set
  - Run a shell script against a z/OS UNIX System Services file

- Create complex remote actions, for example:
  - Invoke an action on multiple hosts
  - Invoke interactive and conditional multi-actions

4.8.3 Maintaining files, menus, and actions

You can use Rational Developer to maintain your files, menus, and actions.

Organization

Menu actions are the components of menus within Rational Developer. You store these actions in menus, which, in turn, are stored in files, as shown in Figure 4-57.

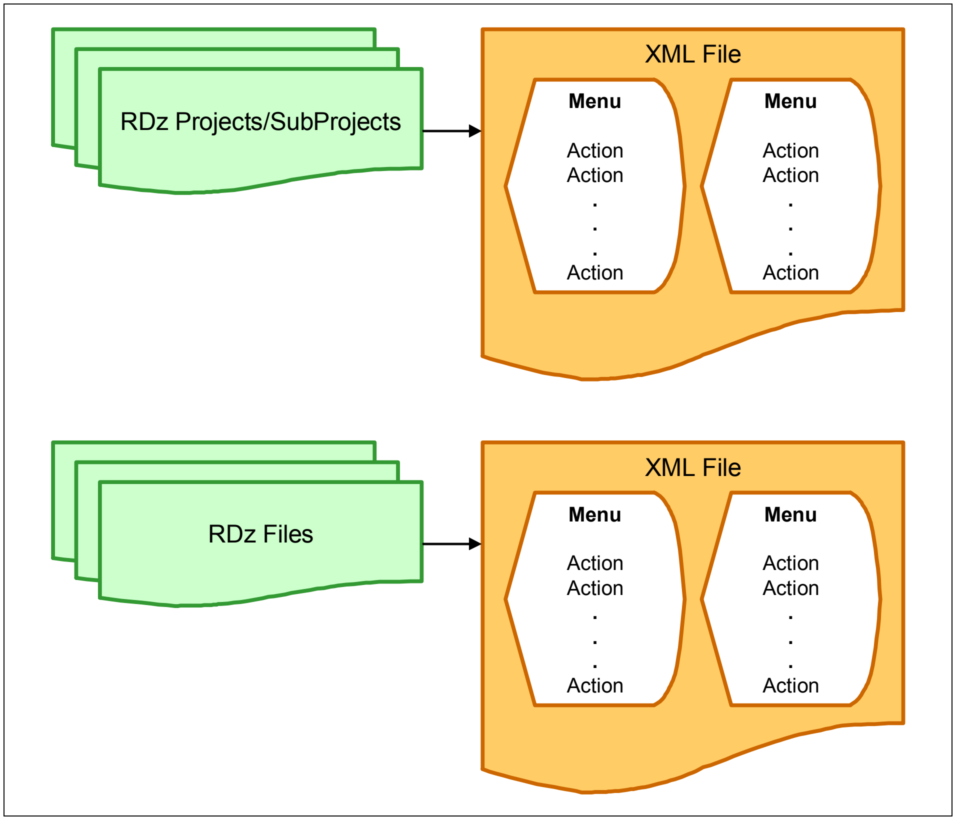


Figure 4-57 Actions are stored in menus that are stored in files

Files

You can create new files in the Menu Manager to store action and menu definitions. You save these files in XML format, which makes them easy to import to other users.

A file can only contain actions and menus for one type of resource. For example, a file that contains actions that act on files cannot also contain actions that act on projects and subprojects. The contents of a file, such as action definitions and menu definitions, appear in the Actions and Menus tabs within the Actions and Menus preferences page in the Menu Manager.

Menus

A pop-up menu, in the Rational Developer context, opens when you right-click a project, subproject, or a file. You construct each menu from the following elements:

- Actions: For performing a predefined action
- Separators: Visually separate groups of actions
- Cascaded submenus: Provide a second-level menu when selected from the parent menu

Each pop-up menu consists of base, custom, or generic menus.

Base menus

A *base menu* contains general actions that apply to either all projects and subprojects or to all files. You can only use two base menus: One base menu for projects and subprojects and one base menu for files. The base portion of a project or subproject pop-up menu is identical for all projects and subprojects, and the base portion of a file pop-up menu is identical for all files. If a base menu is set, the base menu always appears in a pop-up menu, regardless of whether a project, subproject, or file has a custom menu assigned to it.

Custom menus

A *custom menu* contains actions that apply to projects and subprojects or to files. The custom portion of a pop-up menu can differ for each project, subproject, and file. You can use the Menu Selection preferences page to specify custom menus for projects, subprojects, and files.

You can specify the menu for the custom portion of a pop-up menu by using one of the following methods:

- Default menu: You can specify one menu that is used by default for all files and another menu that is used by default for all projects and subprojects. All files use the same default menu, and all projects and subprojects use the same default menu. Default menus only appear in a pop-up menu when a project, subproject, or file does not have a custom menu assigned to it. If this is the case, Rational Developer uses the default menu that is specified in the Menu Selection preferences page.
- Custom menu: You can assign a custom menu to individual projects, subprojects, and files. You can also assign the same custom menu to all files within a specific project or subproject. This capability is useful if you use a variety of projects or subprojects for separate purposes and want separate actions to appear on the pop-up menu for each project or subproject.

Generic menus

When you create a new action using the Menu Manager, you can select the "Show on generic menu" attribute to indicate that you want the action that you are creating to appear on the generic menu. All these marked actions are combined automatically at run time when you right-click a resource to create the generic menu.

You can specify the contents of pop-up menus using the Menu Selection preferences page. This page is available in the workbench by selecting **Window** → **Preferences** to open the Preferences page. In the left navigation pane, double-click the **Menu Manager** node to expand it. From the Menu Manager tree, select **Menu Selection** to open the Menu Selection preferences page. The Menu Selection preferences page is populated with a list of the menus that are available in the Menu Manager.

Actions

You can execute the actions that appear on pop-up menus for files, projects, and subprojects using several methods. For example, you run certain actions using a command on a remote system. You run other actions using local Java classes that operate on selected files, projects, or subprojects.

You can use the Menu Manager Preferences page to contribute new custom actions for files, projects, and subprojects to appear on pop-up menus. When you create a new action in the Menu Manager, you must specify the type of resource on which this action acts.

> **Tip:** Actions that are created in the Menu Manager can only operate on either file resources or project and subproject resources.

You can create the following types of new actions:

- *Simple local*: You can define a simple local action by performing one of the following tasks:
  - Defining the command string that is run when the action is selected from a pop-up menu
  - Defining the Java class to run when the action is selected from a pop-up menu. To define the Java class to run when an action is selected from a pop-up menu, you must write Java code and contribute it to the Menu Manager plug-in through an extension point.
- *Simple remote*: You can define a single command string for each simple remote action that you create. When a simple remote action is selected from a pop-up menu, the command string that you defined for this action runs.
- *Complex remote:* You can create the following types of complex remote actions by stringing simple remote actions together:
  - *Multisystem script*: A multisystem script action runs multiple simple remote actions on separate remote hosts. You can create a multisystem script action by selecting the simple remote actions that you want to run on a variety of remote hosts, and associating host information, such as the host name, relative path, and user name with each of these actions.
  - *Interactive*: You can create interactive actions by connecting simple remote actions together using condition objects. *Condition objects* specify a condition (placed on the return code of an action) that must be satisfied for subsequent actions to run. For example, a simple remote action to find the test.txt file might have a condition object that specifies that the file must be located before subsequent actions can run. Satisfying this condition invokes the next action to open this file in Notepad.

Rules for general actions

You can create local actions for files, projects, and subprojects. You can create remote actions only for files.

Before you create a new custom action, you must decide the type of action that you want to create, based on the following criteria:

- Each action in the Menu Manager can only apply to either files or projects and subprojects, but not to both. If you want your new action to appear on pop-up menus for files, create a new file action. If you want the action to appear on project and subproject pop-up menus, create a new project/subproject action.
- If you want to run the new action locally, create a local action. To run the new action remotely, create a remote action.

- Determine the purpose of the new action. For example, if the action that you want to create is a simple action that runs a specific command on a command-line interface, create a simple action. If the action that you want to create requires complicated processing and graphical user interface (GUI) prompts, you might want to create a Java class to run this action and, then, create a simple local action to run the Java action.
- If your action requires that multiple commands will be issued, create a multisystem script file action.

4.8.4 Creating a simple remote file action in a step-by-step guide

In this topic, we demonstrate, in a few steps, how you can implement a simple TSO REXX as a menu action.

Consider a REXX exec that accepts a parameter from the user, reads a sequential data set, and shows data on the terminal. For our example, we use the REXX that is shown in Example 4-2.

Example 4-2 REXX exec

```
/* REXX - GETEQUIP Example */
ARG CATEGORY .

ADDRESS TSO "ALLOC FI(T) DA('RTPOT03.EQUIP') SHR REU"
ADDRESS TSO "EXECIO * DISKR T (STEM IN. FINIS"
ADDRESS TSO "FREE FI(T)"

DO I = 1 TO IN.0
  IF SUBSTR(IN.I,9,4) = CATEGORY THEN
    SAY 'Equipment ID:' SUBSTR(IN.I,1,8)
END I

EXIT
```

To invoke this REXX from your Rational Developer menu, complete the following steps:

1. In the workbench, select **Window** → **Preferences** to open the Preferences page.
2. In the left navigation pane, double-click the **Menu Manager** node to expand it.
3. From the Menu Manager tree, select **Actions and Menus** to open the Actions and Menus preferences page (Figure 4-58).

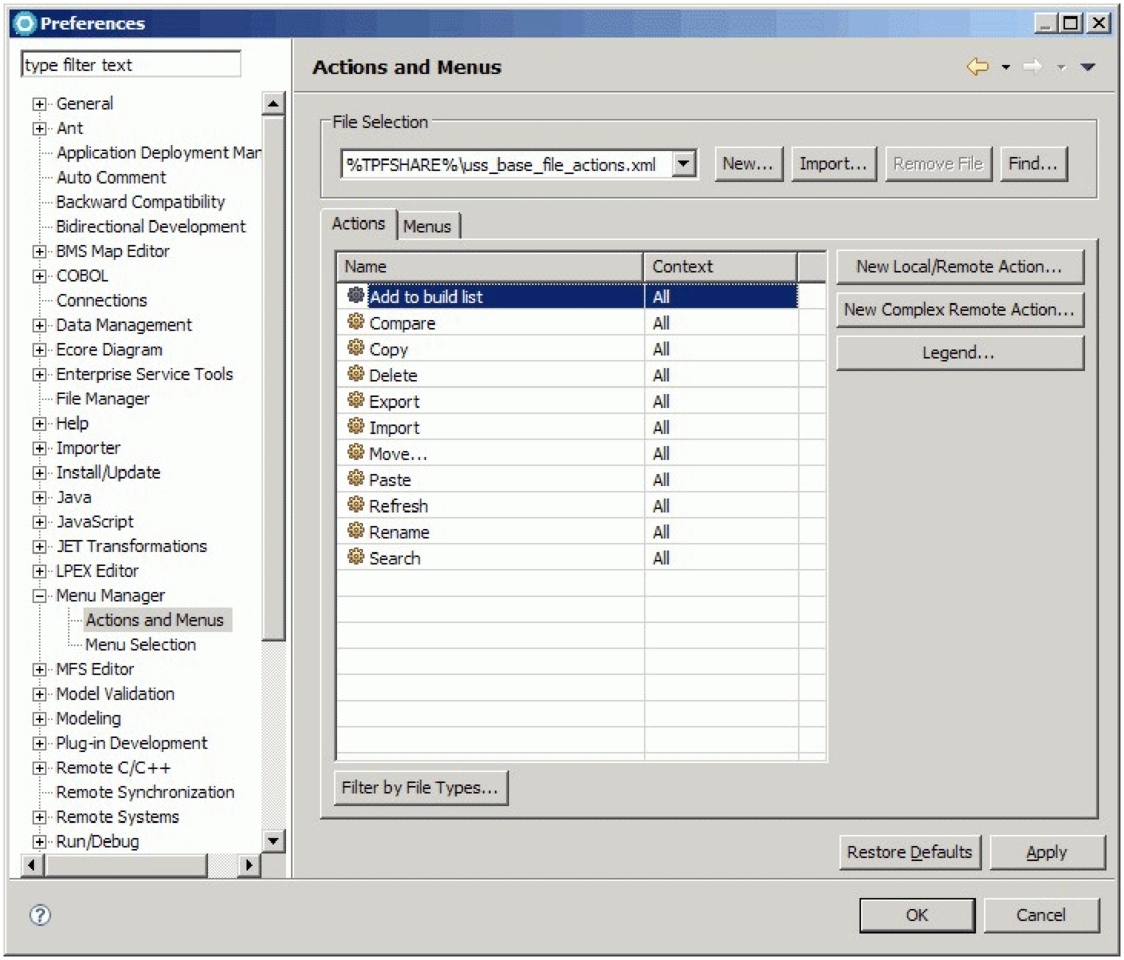


Figure 4-58 Actions and Menus preferences page

4. In the File Selection drop-down list (Figure 4-58), select the **uss_base_file_actions.xml** file.
5. In the Actions tab, click **New Local/Remote Action** to open the Menu Manager New Action Wizard page (Figure 4-59):
   – In the Name field, type the name of the new file action that you want to appear in the pop-up menu.
   – In the Comment field, type a short description that you want to appear as ToolTip text when you hover over this action in the pop-up menu.
   – In the Context field, select the context in which this action is visible. For our example, we select **All**.
   – Click **Next** to proceed to the File Associations page.
6. Select `*.*` from the Available File Types drop-down list, and click **Add** to select all files that contain a file name extension.

> **File types:** The simple local file action that you create only appears on the pop-up menu if each of the selected files corresponds to an associated file type that is specified in the File Associations page.

7. Optional. To add a new file type to the list of file types that appears in the File Associations page, click **File Types** to open the File Type Manager window. In the Existing Types section, click **Add** to open the New File Type Information window. In the Enter File Extension field, type the file name extension for the new file type that you want to add to the list of existing file types, and click **OK** to add it to this list. Click **Finish** to add the new file type to the list of associated file types. Click **Next** to proceed to the Run Options page (Figure 4-59).

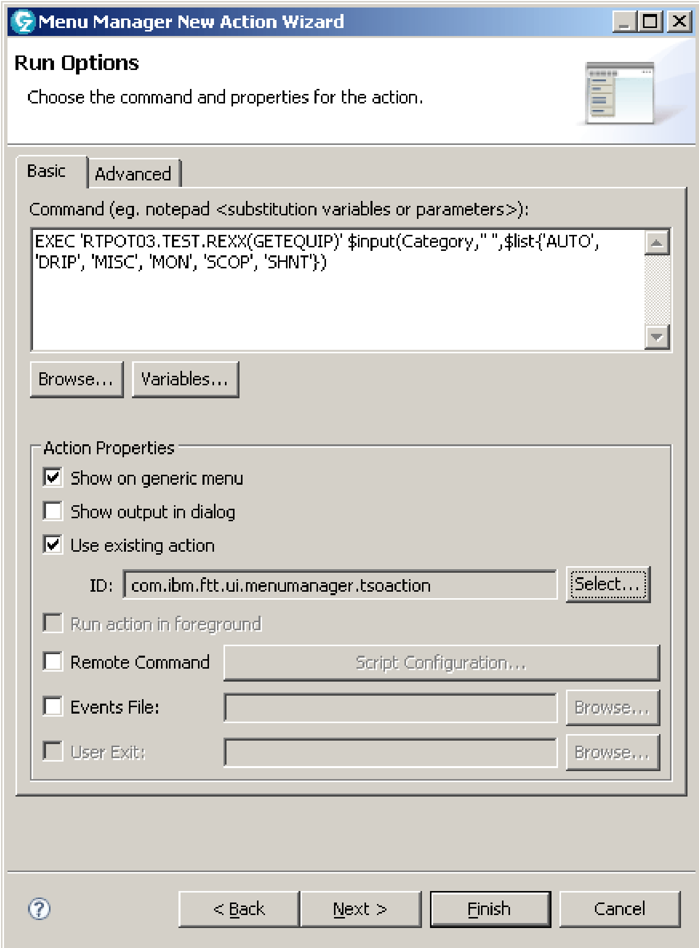


Figure 4-59 Run Options page

8. In the Basic tab, in the Command field, type the following command string:

```
EXEC 'RTPOT03.TEST.REXX(GETEQUIP)' $input(Category," ",$list{'AUTO',
'DRIP', 'MISC', 'MON', 'SCOP', 'SHNT'})
```

   Here is an explanation of the string components:

   – EXEC is the standard TSO EXEC command, which is used to execute a REXX exec. Here, we fully qualify the REXX member, but you can use shorter forms if your REXX library is concatenated to SYSPROC in the ISPF.conf.

   – The **`$input`** keyword notifies Menu Manager to accept input from the user before executing the action.

– The `$list` keyword provides the option of including a list of values from which the user can select a value.

– The following example is the syntax of the `$input` command: **`$input(PromptString, DefaultValue, $list{string1,string2,string3, . . .,stringn})`**, which we define:

- **`PromptString`**
  Specify the dialog box label that prompts the user for input. The prompt string can contain spaces but no commas (,) or dollar signs ($). The prompt string is required.
- **`DefaultValue`**
  Specify the value to use as the default if the user does not enter a value when prompted. The default value can contain spaces but no commas (,) or dollars signs ($). The default value is required.
- **`$list{string1,string2,string3, . . .,stringn}`**
  Specify a list of strings from which the user can select a value. This variable is optional. If you include it, the dialog prompts the user to select one of these strings from a drop-down list. If you omit it, the dialog prompts the user to type a value. Each value in the list can contain spaces but no commas (,) or dollar signs ($).

9. In the Action Properties section, you can select these options:

– Optional. To display the file action on the generic file menu, select **Show on generic menu**.

> **Show on generic menu:** If you do not select the **Show on generic menu** check box, you must add the file action to a menu and attach that menu to a file before the action appears. If you select the **Show on generic menu** check box, the file action appears on the custom portion of the pop-up menu for all files that use the generic file menu.

– Select **Show output in dialog** to specify that you want output from the action to display in a dialog box instead of in the Console.

– Select **Use existing action** to use an action implemented as a Java class. Click **Select** to open the Action Selection dialog box, and select **com.ibm.ftt.ui.menumanager.tsoaction**.

10. Click **Next** to proceed to the Add Actions to Menus page (Figure 4-60). Select **Add to menus**, and then, select the check box for each menu to which you want to add the new remote local file action.

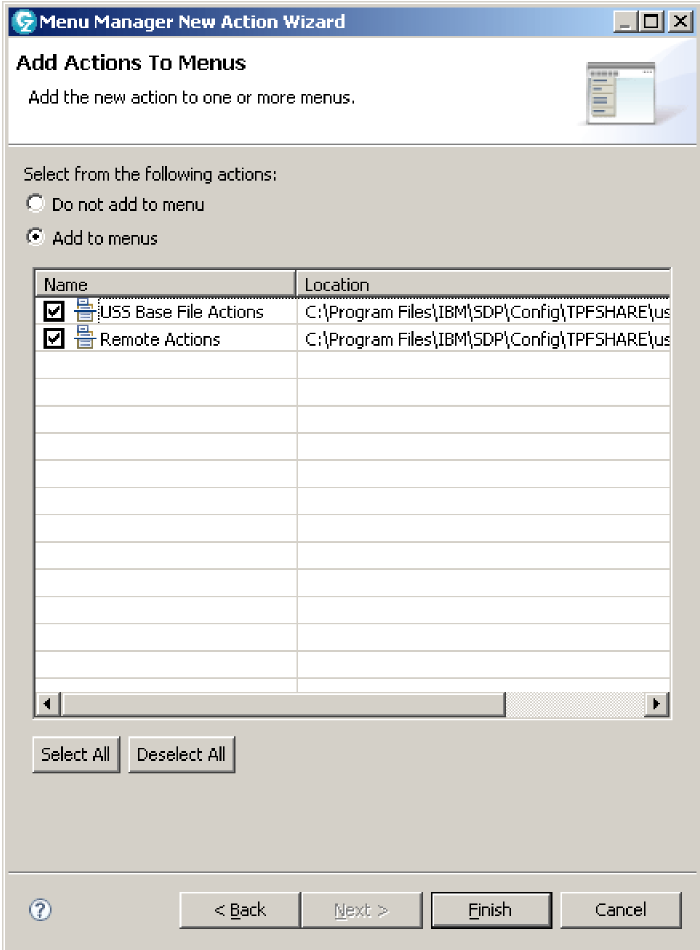


Figure 4-60 Add Actions to Menus

To change your selections, click **Back**. To create the simple remote file action, click **Finish**. The file action appears in the list of actions in the Actions tab.

Click **OK** to add the new simple remote file action to the selected file.

4.8.5 Creating simple local file action in a step-by-step guide

You can use the Menu Manager Actions and Menus preferences page to create simple local file actions that appear on pop-up menus.

To create a new simple local file action, complete the following steps:

1. In the workbench, select **Window** → **Preferences** to open the Preferences page.
2. In the left navigation pane, double-click the **Menu Manager** node to expand it.
3. From the Menu Manager tree, select **Actions and Menus** to open the Actions and Menus preferences page (Figure 4-61).

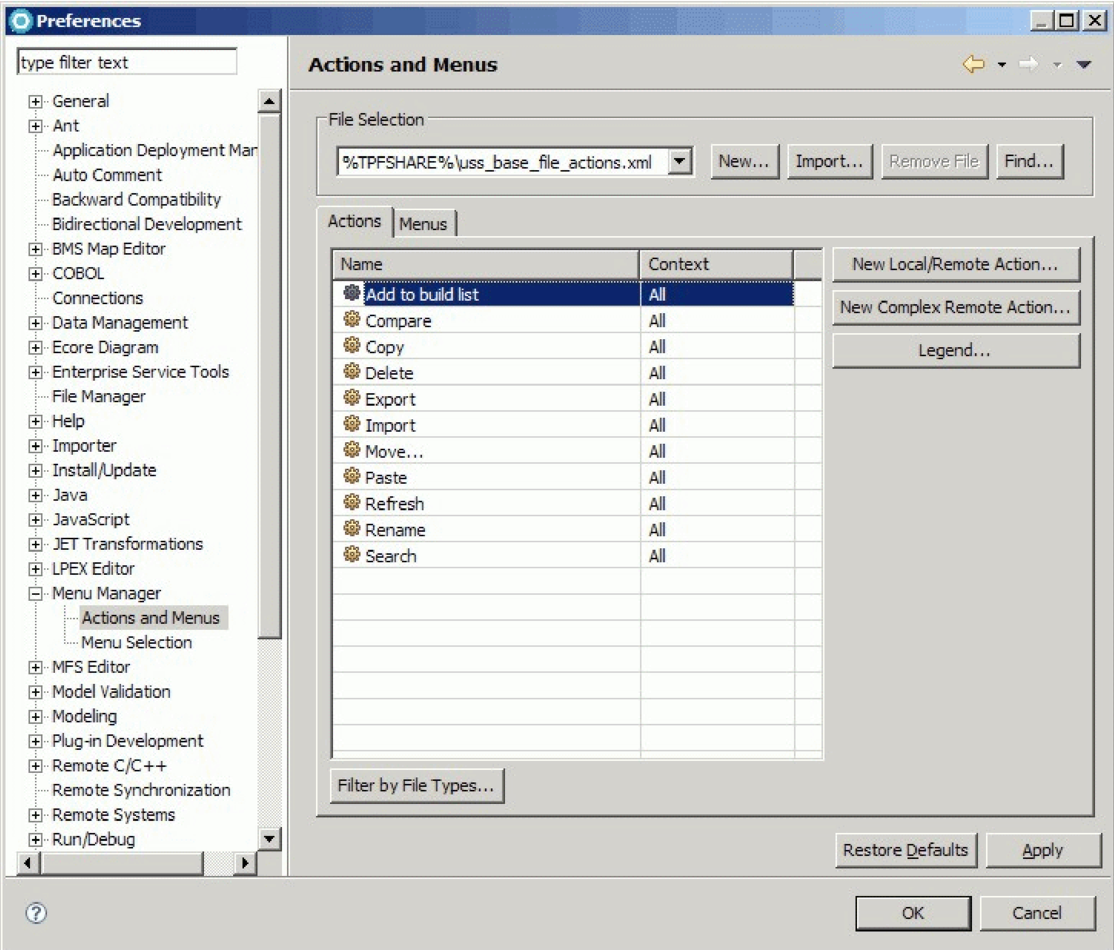


Figure 4-61 Actions and Menus preferences page

4. In Figure 4-61, perform one of these steps:
   - From the File Selection drop-down list, select the file in which you want to create the new file action.
   - Click **New** to create a new file in which to create the new file action.

- Click **Import** to import an existing file in which to create the new file action.

  Tip: You can only create a file action in a file that contains action and menu definitions that apply to files. The icon beside the File Selection drop-down list indicates if the file contains action and menu definitions that apply to projects and subprojects or files.

5. In the Actions tab, click **New Local/Remote Action** to open the Menu Manager New Action Wizard:
   - In the Name field, type the name of the new file action that you want to appear in the pop-up menu.
   - In the Comment field, type a short description that you want to appear as ToolTip text when you hover over this action in the pop-up menu.
   - In the Context field, select the context in which this action is visible. For example, if the z/OS UNIX Subprojects context is chosen, this action only shows on z/OS UNIX System Services files. Depending on the context that is chosen, the properties that are available on subsequent pages of the new action wizard might differ.
6. Click **Next** to proceed to the File Associations page. Select the file types to associate with this file action by performing one of these tasks:
   - Click **Add All** to select all the listed file types.
   - Select the file type that you want to associate with the file action, and click **Add**.
   - Select ***.***. Click **Add** to select all files that contain a file name extension.

     Tip: The simple local file action that you create only appears on the pop-up menu if each of the selected files corresponds to an associated file type that is specified in the File Associations page.

   Optional. To add a new file type to the list of file types that appears in the File Associations page, click **File Types** to open the File Type Manager dialog box. In the Existing Types section, click **Add** to open the New File Type Information dialog box. In the Enter File Extension field, type the file name extension for the new file type that you want to add to the list of existing file types, and then, click **OK** to add it to this list. Click **Finish** to add the new file type to the list of associated file types.

   Optional. To remove a file type from the list of associated file types, click **File Types** to open the File Type Manager dialog box. In the Existing Types section, select a file type, click **Remove**, and then, click **Finish** to delete it from the list of files to associate with this file action.

7. Click **Next** to proceed to the Run Options page (Figure 4-62).

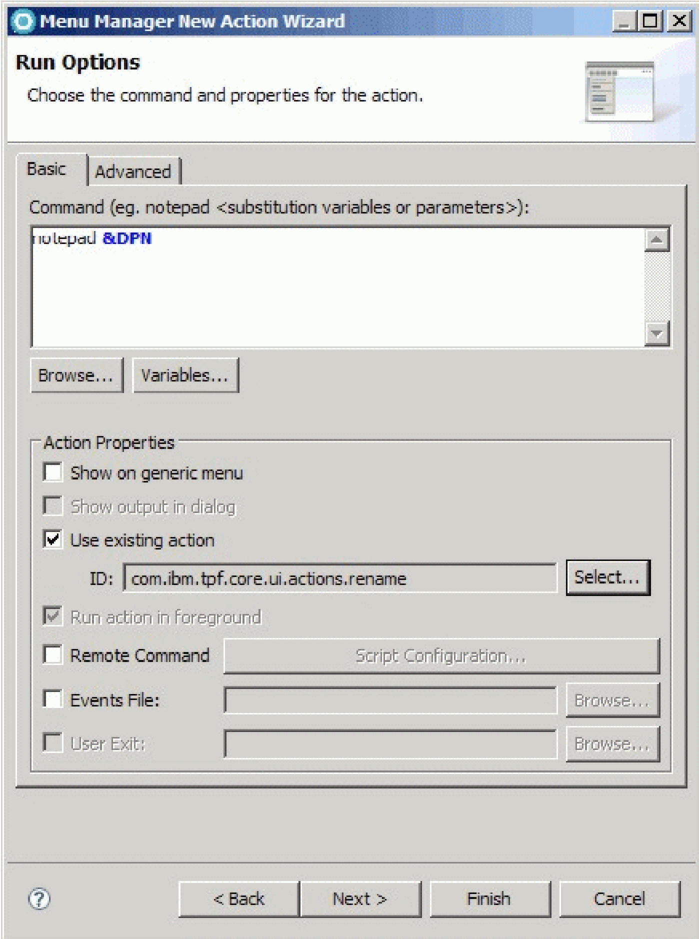


Figure 4-62 Run Options page

8. In Figure 4-62, in the Basic tab, perform these tasks:

   Optional. To specify the command string that is run when this file action is selected from a pop-up menu, perform one of the following steps:

   – In the Command field, type the command string.

   – Click **Browse** to browse for the location of a file on the local system that you want to include in the command string. The Browse button is disabled when the Remote Command check box is selected.

   – Click **Variables** to open the Substitution Variables dialog box. Select the variable that you want to add to the command string, click **Insert**, and then, click **Close** to close the dialog box. The substitution variable appears in the Command field.

> **Important:** When selecting substitution variables to add to the command string, ensure that you select variables from the context in which the simple local file action appears on the pop-up menu.

In the Action Properties section:

Optional. In the z/OS UNIX Subprojects context, follow these steps to reference an existing IAction:

- Select **Use existing action** to use an action implemented as a Java class.
- Click **Select** to open the Action Selection dialog box.
- From the list of actions, select the ID of the IAction class that is invoked for this action. This list of actions is populated with actions that are contributed using the com.ibm.tpf.menumanager.actions extension point, for example, com.ibm.tpf.core.make.ui.actions.LoadTPFAction.
- Click **OK**. The selected action shows in the ID field. If you have not already specified the command string that is run when this project/subproject action is selected from a pop-up menu, you must select **Use existing action**.
- Optional. To display the file action on the generic file menu, select **Show on generic menu**. If you do not select the Show on generic menu check box, you must add the file action to a menu and attach that menu to a file before the action appears. If you select the Show on generic menu check box, the file action appears on the custom portion of the pop-up menu for all files that use the generic file menu.
- Optional. In the z/OS UNIX Subprojects context, to specify that you want the action to run in the foreground, select **Run action in foreground**. If you select the check box, the action runs in the foreground and the main GUI is locked. To unlock the GUI, click **Cancel**.

> **Tip**: If the action references an existing IAction, this check box is disabled. The IAction determines if the action runs in the foreground.

- Optional. To specify that you want output from the action to be displayed in a dialog box instead of in the Console, select **Show output in dialog**.
- Optional. In the z/OS UNIX Subprojects context, select **Events File** and click **Browse** to browse for the location of the events file. The contents of the events file appear in the Remote Error List view.

9. In the Advanced tab, perform these tasks:

   Optional. By default, messages that are generated when an action runs are appended to existing messages that appear in the Console view. In the Clear console section, specify if you want to clear the console before running this file action:

   – To use the default setting that is specified on the Menu Manager Preferences page, click **Use preference setting**.

   – To override the default setting that is specified on the Menu Manager Preferences page, click **Use custom setting**. To set your custom setting to clear the console before running the file action, select **Clear console before running**.

   In the z/OS UNIX Subprojects context, in the Additional action properties section, perform these tasks:

   – Select **Refresh enclosing projects on action completion** to refresh the project/subproject that contains the project resource on which this action is acting, when the action completes. The project is refreshed regardless of whether the action completes successfully or fails.

   – Optional. To collect all the file names and to run the command one time for the set of selected files, select **Collect Names**. Or, to run the command one time for each selected file, clear the Collect Names check box.

   – Optional. To verify that the length of the command does not exceed the specified limit, select **Length Limit**. Type the numeric value of the maximum allowable command length in the Length Limit field. If the command exceeds the maximum length, it is not run. If you do not select the Length Limit check box, the length of the command is unlimited.

   Click **Next** to proceed to the Add Actions to Menus page.

10. In the Add Actions to Menus page, you can perform these tasks:

   – Optional. Click **Add to menus**, and then, select the check box for each menu to which you want to add the new simple local file action.

   – To change your selections, click **Back**. To create the simple local file action, click **Finish**. The file action appears in the list of actions in the Actions tab.

   – Click **OK** to add the new simple local file action to the selected file.

4.8.6 Summary

We have described the following topics in this section:

- The purpose, concept, and importance of Menu Manager within the Rational Developer toolbox
- The use of Menu Manager in assisting you with your daily work
- The structure of Menu Manager components and the interaction of each component with other components and with the user
- The ability to create, import, and alter menus
- The ability to attach menus to projects, subprojects, and files
- The ability to create simple local and remote actions

4.9 Rational Developer for ISPF developers

Well over 95% of Rational Developer users either began with ISPF and are advanced ISPF developers or have used ISPF to perform production development work in their careers. So, getting from ISPF to Rational Developer is not difficult.

The more time that you have invested in green-screen tools, the more you will benefit from the following Rational Developer characteristics:

- An understanding of the parallels between ISPF and Rational Developer development:
  - Research the documents on the IBM COBOL Cafe:

    `http://www-949.ibm.com/software/rational/cafe/community/cobol`
  - When inside the COBOL Cafe, search on `ISPF Rational Developer`.
  - See Figure 4-63 for an example of the kind of information that is helpful in making this transition.

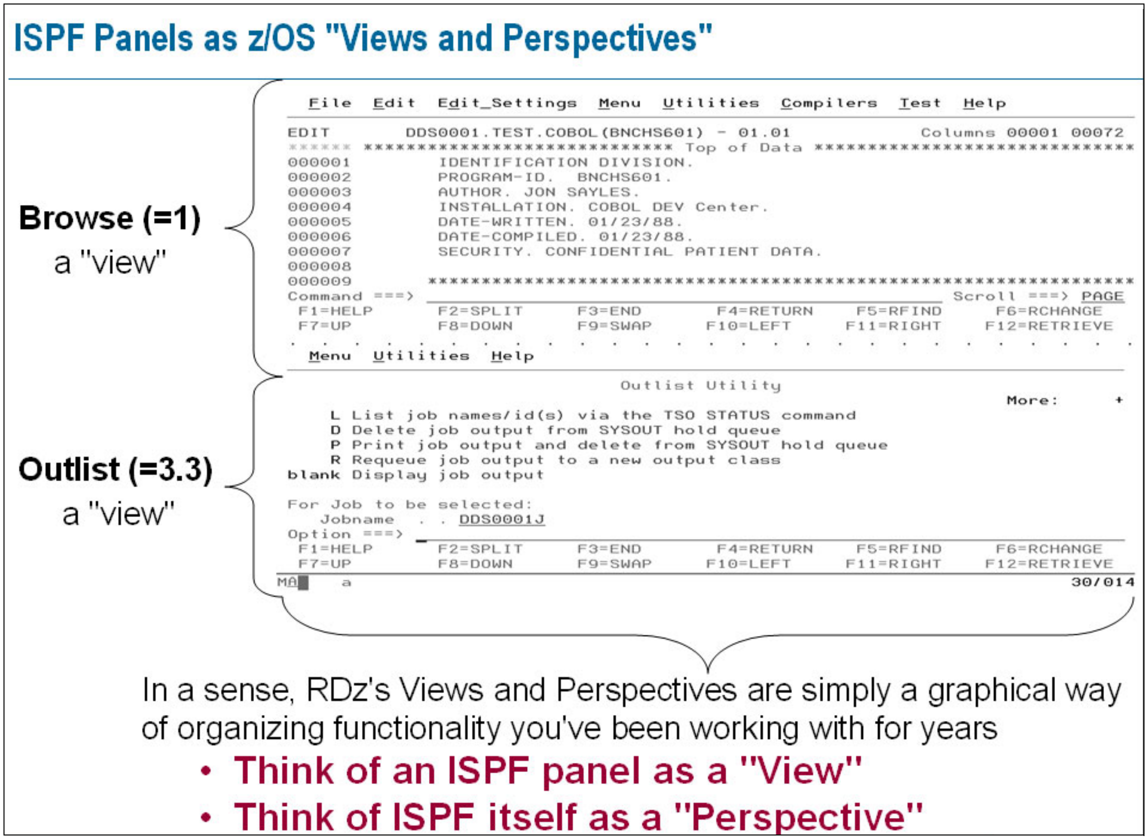


Figure 4-63 Materials from Rational Developer for ISPF developers

- Compare and contrast the ISPF editor functionality with the Rational Developer LPEX editor. Documents on the COBOL Cafe contain functional matrixes.
- Use the ISPF editor profile, by selecting **Windows** → **Preferences** → **LPEX** and set your profile to ISPF.
- It takes time to change to Rational Developer. Even using the Rational Developer ISPF editor profile, ISPF developers see differences in the following areas:
  - Missing ISPF commands
  - New editor behaviors
  - New geometric proportions

  Learning to substitute the mouse for typed navigation commands can take time to internalize and make intuitive.

- Mentoring and support during the initial learning phase:
  - Extremely beneficial when first starting out (whether in class or through self-study) that you have access to someone who knows Rational Developer
  - Useful when you get stuck and are tempted to return to ISPF to meet a deadline
  - Helpful to have someone in person to remind you, “No, it is the other right mouse button to get to the context menu.”

As shown in Appendix A, “IBM Rational Developer for System z productivity benchmark comparison” on page 587, which describes recent deep-dive studies by IBM and clients, the productivity efficiencies and benefits of Rational Developer versus ISPF are considerable and compelling. Studies show an overall average of 30% less time to complete tasks using Rational Developer.

4.10 Summary

In this section, we have introduced you to Rational Developer. At this point, you understand the following foundation Rational Developer terms and concepts:

- Rational Developer
- Important Rational Developer views and perspectives
- Rational Developer menuing system
- Rational Developer LPEX editor and its ISPF profile
- Rational Developer integration:
  - DB2
  - IMS
  - CICS
  - PD Tools and specifically:
    - Debug Tool
    - File Manager
    - Fault Analyzer
  - Rational Asset Analyzer
  - Your mainframe REXX (through Menu Manager)
  - Your shop’s source control management system through CARMA Menu Manager or another solution

Hopefully, you have seen how it is possible to map these products back to ISPF functionality and understand why Rational Developer technology can make your job easier and more fun.

We have addressed the ISPF tools that you use on your z/OS system today and described the coverage, equivalents (Figure 4-64), and extensions to the development functionality that is provided by the EM Workbench (Rational Developer and Rational Asset Analyzer integration/Rational Team Concert for System z), as shown in Figure 4-65.

EM Workbench encapsulates all the areas of tooling, and many additional functional facilities exist to help you:

- Get your job done faster
- Move you past the annoying daily tasks that typify too much of mainframe development work
- Significantly decrease the amount of time that you spend typing to get around in your code and in the tools. Features, such as Content Assist, decrease the time that you spend writing code and compiling it to uncover syntax errors.
- See your application more clearly and more in-depth using Rational Asset Analyzer integration and the optimized windows

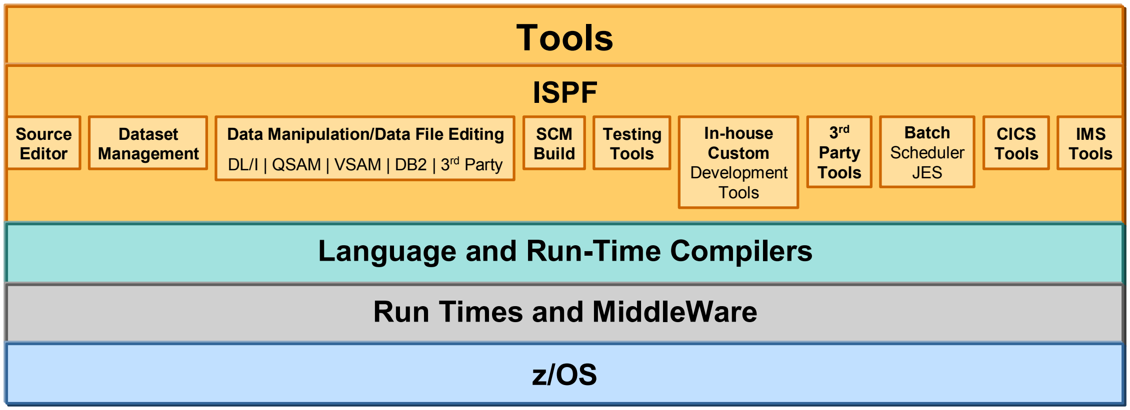


Figure 4-64 TSO and ISPF traditional tools

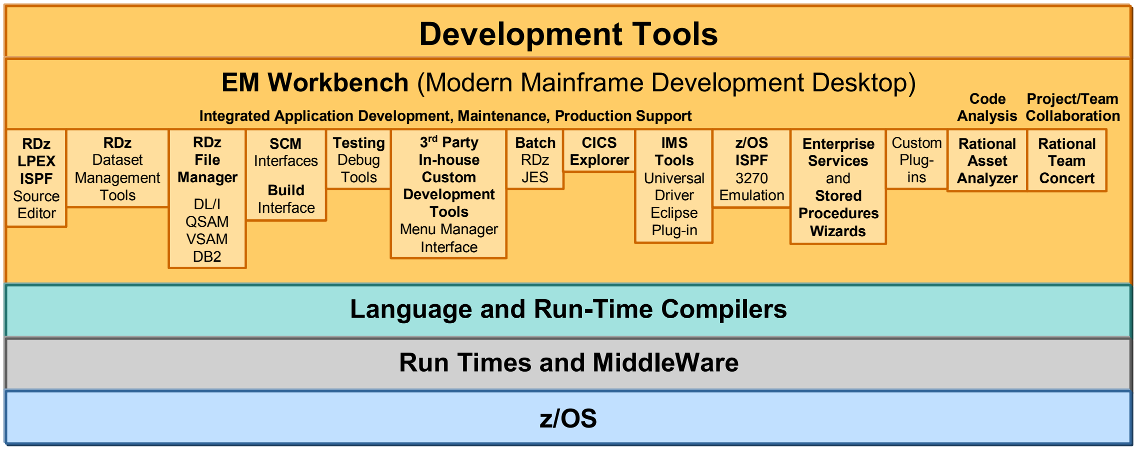


Figure 4-65 Integrated tools of the EM Workbench, representing the optimized life cycle

5

Rational Asset Analyzer

This chapter describes Rational Asset Analyzer Version 5.5 and its features, functionality, terms, and vocabulary from the vantage point of a z/OS ISPF developer.

Rational Asset Analyzer is a static code analysis tool of the Enterprise Modernization (EM) Workbench. Rational Asset Analyzer is used to improve the productivity of project managers, lead technical developers, application and business analysts, and application programmers.

Rational Asset Analyzer is a prominent part of the optimized life cycle, because source code analysis tools play a central role in increasing a development team's efficiency. Rational Asset Analyzer improves the output of all development stakeholders and helps you deliver correctly maintained, working application code faster and more predictably than with manual approaches.

Rational Asset Analyzer helps developers find and fix problems across most of the steps in the development process. Unlike dynamic code analysis, static code analysis requires no test cases and can analyze and find defects even in nonoperational code that is still under development.

Rational Asset Analyzer's key value propositions include but are not limited to these benefits:

- Accelerated project delivery in response to business drivers
- Reduced risk through better understood code semantics
- Increased productivity
- Improved quality of application changes
- Greater intellectual control of applications
- Improved project management decisions
- Shortened time-to-market of application changes, modifications, business rule mining, and service-oriented architecture (SOA) reengineering

In this chapter, we describe the following topics:

- Rational Asset Analyzer and how it analyzes static code
- Terms and concepts
- The process of loading inventory into the Rational Asset Analyzer repository
- The use of Rational Asset Analyzer by various stakeholders
- The four primary use cases for Rational Asset Analyzer:
  - Project management
  - Application understanding
  - Application analysis, including Impact Analysis
  - Application documentation

We also describe briefly how to extend Rational Asset Analyzer's in-stream capabilities through custom analysis queries.

5.1 Static Code Analysis

Static Code Analysis is the investigation of application software behavior and application semantics performed by reading source code without actually running or debugging programs that were built from that software. In Chapter 2, “The optimized life cycle: Project managers and implementation teams” on page 15, we performed a Dynamic Code Analysis while debugging programs. We used the IBM Debug Tool with IBM Rational Developer for System z for an example of Dynamic Code Analysis.

You can use the Static Code Analysis tool to understand an application’s *metadata* or software behavior and semantics. Using the Static Code Analysis tool allows you to make better decisions about how to code, change, modernize, or fix an application.

5.1.1 Metadata: Elements and dependencies

The outcome or result of Static Code Analysis is essentially an understanding or “mental map” of elements, their attributes, and their dependencies:

- Significant application elements are elements of interest to your tasks or research. In z/OS applications, elements consist of these objects:
  - Transactions
  - Batch jobs
  - Programs and all of the program sub-elements:
    - Paragraphs and sections
    - Variables
    - Literals
  - Copybooks
  - DB2 tables, files, and databases
  - Stored procedures
- Attributes of the application elements: All programs and copybooks have variables, functions or paragraphs, and literals. Batch jobs have steps, and steps have data definition (DD) statements that refer to files. DB2 tables consist of columns. It is relatively easy to reconstitute a table declaration from the DB2 table Data Definition Language (DDL). However, programs with thousands of variables and batch jobs with hundreds of files are more quickly and easily grasped using tools that inventory, classify, and provide hyperlinked browse-enabled lists of the attributes.

- The dependencies among elements are the key piece of metadata when it comes to maintenance and support tasks. Given the size, complexity, and scope of production assets, understanding the dependencies of the significant application elements can take a long time.

By understanding the metadata about an application, we can answer questions:

- Which programs use this database
- Which databases, files, and tables are accessed by a program
- Which programs comprise an IMS or CICS transaction and what is the called/calling sequence
- Which files are accessed by which batch jobs and in which steps
- Which paragraphs are performed in a program and what is the hierarchical structuring
- Application "slicing" questions
- If you change a certain element (database column, field, variable, file, or set of program statements), what downstream application elements are affected by the change

That final question is known as "*impact analysis*" and often lies at the core of z/OS application maintenance and even production support work.

5.1.2 Static Code Analysis: Scope

In most cases, you perform Static Code Analysis on a certain release of the application. However, the scope of Static Code Analysis can differ:

- The scope can be much larger than one application. Rational Asset Analyzer supports Enterprise-wide Static Code Analysis that you can use to capture the intra-application and inter-application metadata.
- The scope can be smaller. Often, you are interested in more granular metadata, such as a single program and its control or data flow or a file, copybook, variable, or literal. In these cases, you must factor out or filter out information that is irrelevant to the current task.

5.1.3 Why Static Code Analysis

Essentially, you cannot modify a single line of an existing production application without performing Static Code Analysis to understand what effect your change will have on the program, file, batch job, or transaction. It also helps you to know what effect your change will have on downstream executables and dependent system elements.

Dynamic Code Analysis can provide a subset of the information that you need to maintain an application with confidence, but a subset is never sufficient. Think of it this way:

- In all cases of maintenance and support work, you perform Static Code Analysis
- In specific cases, you augment your Static Code Analysis with Dynamic Code Analysis

5.1.4 Why Rational Asset Analyzer for Static Code Analysis

Historically, developers performed Static Code Analysis by reading source listings (job control language (JCL) and programs), penciling notes, turning down the pages on listings, and adding comments on sticky notes. They also performed ISPF 3.4 searches and read code while they were logged in to Time Sharing Option (TSO).

If that method sounds familiar, you probably still perform Static Code Analysis this way. However, if you have a tool that builds the application metadata automatically, you can use it to help you:

- Jump-start your work: Rational Asset Analyzer produces analysis information in seconds compared to the days it takes to produce it manually.
- Double-check your work: Rational Asset Analyzer does not get tired, and it will not make assumptions and will not miss situations, circumstances, elements, and dependencies that even the best programmer/analyst might miss.

Now, we discuss how Rational Asset Analyzer works and what it delivers.

5.2 What is Rational Asset Analyzer

Rational Asset Analyzer is a browser-based Static Code Analysis tool that offers an efficient way to perform application analysis. It assists in the development, maintenance, testing, and support of life-cycle phases to identify effects across the enterprise level or application level or at a lower level of granularity. Rational Asset Analyzer identifies and catalogs elements and element relationships, gathers metrics related to the IT artifacts, and also provides a browser-based interface to search, explore, and report on the gathered information.

5.2.1 Capabilities

Rational Asset Analyzer has the capability to programmatically collect information about applications from the places where their source is stored. It organizes the metadata (information about source artifacts) into a single repository that is known as a "*metadata repository*". Because scanning artifacts and processing gathered data are automated, Rational Asset Analyzer can significantly reduce the time that we spend to research application-related questions. It can help ensure a more complete analysis with greater accuracy and reduced risk. You can augment the metadata as well with human-scribed semantic information, which allows you wide latitude in terms of customizing Rational Asset Analyzer and deepening its value across an organization.

5.2.2 Scenarios

Next, we have listed several practical scenarios where Rational Asset Analyzer is used by clients worldwide.

End-to-end analysis

- At a high-level of systems analysis, Rational Asset Analyzer identifies relationships that span software systems and even separate platforms or environments.
- For programmer and programmer/analyst tasks, Rational Asset Analyzer identifies the calling hierarchy, paragraph flow, and data passed among software programs.
- For managers, Rational Asset Analyzer provides a set of dashboards to help you understand the composition of a software application that includes various metrics and counts that are broken down by complexity, language, or environment.

Application understanding and documentation

- Rational Asset Analyzer significantly reduces the time that it takes to educate new developers, consultants, or outsourcers about existing application metadata, so that they can start contributing quickly and at a deep production level.
- For applications with limited documentation, Rational Asset Analyzer generates relationship diagrams, structural diagrams, and Impact Analysis lists for the quick understanding and online documentation of the current application's behavior as maintained by the code.

Field expansion and maintenance

- *Field expansion* is an industry phrase that relates to tasks surrounding the critical and common work of preparing to modify one or more application elements (that is, expanding a zip code field from 5 to 9 bytes). Field expansion helps you understand, predict, document, and encapsulate that change so that it does not adversely affect other production logic.
- Field expansion is not limited to variables. You might need to change a transaction or batch job stream or modify a DB2 table or IMS database segment.
- Before making any changes, you need to know the effect of those changes on other parts of the application, which is where Rational Asset Analyzer can help you significantly by performing an automated Impact Analysis. You use the automated Impact Analysis to determine the scope of the change and additional changes (both direct and indirect) to other elements of the application.
- Field expansion support requires an extremely deep and broad understanding of z/OS languages and topologies.
- The tool's underlying parsing algorithms for following variable data flow, which is required for field expansion, must be more complex and sophisticated than simply searching on field names with wildcards. Rational Asset Analyzer's parsing engine, which was initially used during Y2K problem resolution, uses compiler-based data mapping.
- For an example of field expansion, see Chapter 13, "Non-Trivial Change Request use case" on page 545.

Test environment generation

- Rational Asset Analyzer can identify the software application artifacts that are related to test data (data sets, files, batch jobs, transactions, and programs) involved in runtime scenarios.
- Knowing the artifacts helps you to replicate the production environment for application-testing purposes to provide a full-scale, robust test environment.

  For example, many shops use Rational Asset Analyzer to help build lists of files that must be migrated to Linux® machines for use with IBM Rational Developer for System z unit test.
- Rational Asset Analyzer can help you answer the questions, "Which components do I need to test a program?" and "What components are required to build?"

Production support

- Rational Asset Analyzer can assist with data flow analysis and control flow analysis, which form the underpinnings of what is described as a root cause analysis of a production ABEND.

Code cleansing

- Rational Asset Analyzer can identify specific literals that are defined in the source to be changed, both in the PROCEDURE DIVISION and WORKING-STORAGE sections. It offers functionality that is well beyond `grep`, text, or ISPF V3.4 search capabilities, because it can find semantic literals, not simply digits or alphabetical characters.
- Rational Asset Analyzer also has a report for “Dead Code” that identifies statements in a program that will never be executed due to the existing control flow structure.

5.2.3 Languages and environments supported

Rational Asset Analyzer supports the following types of z/OS assets:

- Source code: COBOL (including copybooks), PL/I, assembler, and job control language (JCL)
- CICS online regions and transactions
- IMS subsystems and transactions
- SQL statements (Declare CURSOR, Declare TABLE, DELETE, FETCH, INSERT, Singleton SELECT, and UPDATE)
- DB2 catalog information (columns, stored procedures, DB2 subsystems, DB2 tables, and views)

Rational Asset Analyzer also provides over 40 analyzers that understand the structure and contents of distributed assets, including Java sources, Java bytecode, C++, JavaServer Pages (JSP), HTML, XML, and text files. These domain-specific analyzers examine the structure of physical components, such as files, and logical components, such as Java classes. This book’s scope is limited to z/OS traditional maintenance and support, so we do not describe the distributed areas of Rational Asset Analyzer support.

5.2.4 Rational Asset Analyzer product architecture overview

Rational Asset Analyzer works in a client/server-based architecture where the server components are hosted on WebSphere Application Server and connect to

DB2. The client is usually an Internet browser, such as Mozilla Firefox or Internet Explorer (Figure 5-1).

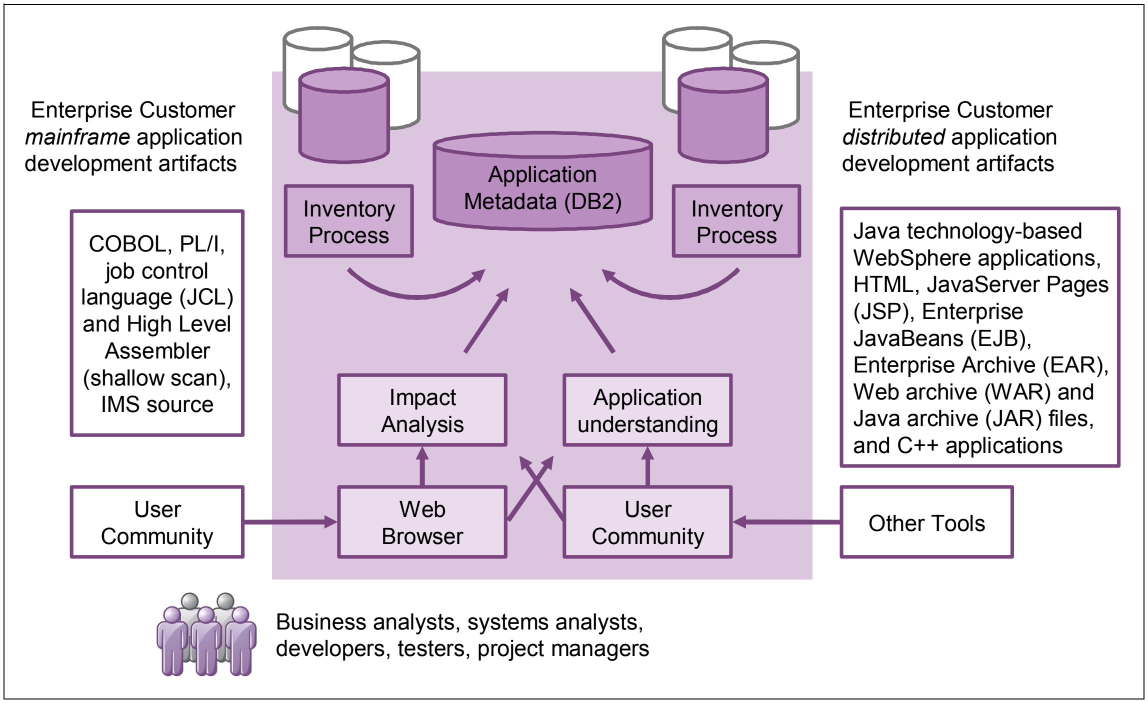


Figure 5-1 Rational Asset Analyzer stakeholders and product architecture

Working with these assets/artifacts involves two major processes.

Inventory process

The *inventory process* is a formal wizard-driven process that consists of these steps:

- Identifying source locations
- Specifying a parsing process
- Executing a parse procedure that scans and analyzes the source files and builds out the metadata repository (Figure 5-2).

While scanning the assets or artifacts, Rational Asset Analyzer (scanners, analyzers, and categorizers) gathers a comprehensive set of information about the assets or artifacts and puts the information into a DB2 database (metadata repository). Note that the product does not store the actual source code in the metadata repository, so you must rescan changed items (changed copybooks and JCL) if you want the snapshot in the product to be up-to-date.

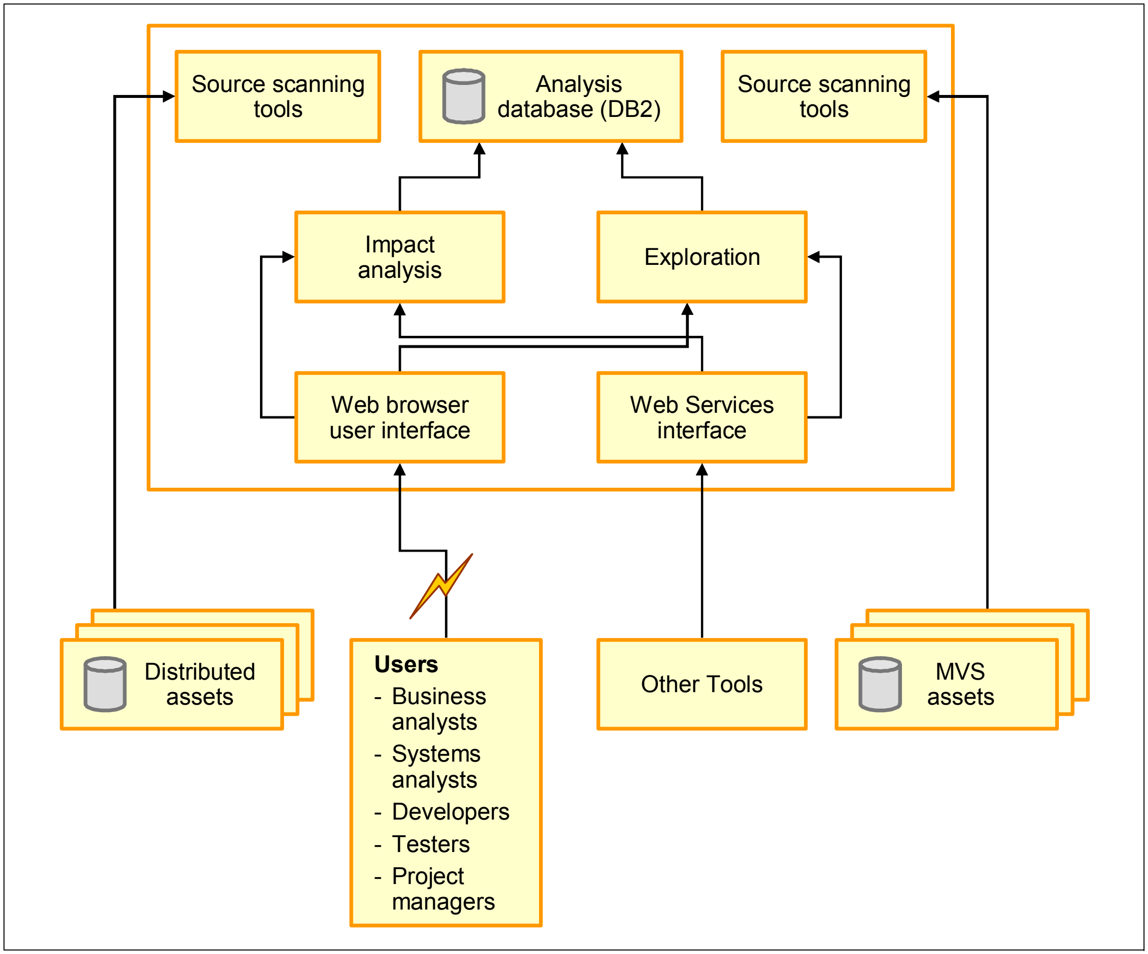


Figure 5-2 Rational Asset Analyzer's asset inventory process: The interface to users is over HTTP (through web pages)

Runtime process

In the runtime process, users use the Web client to search for assets that were identified during the inventory process. We explain the usage model in-depth later in this section.

Rational Asset Analyzer also offers application programming interfaces (APIs) and Representational State Transfer (REST)-based interfaces to integrate it with other third-party tools that might benefit from access to your application or the enterprise application's metamodel.

5.2.5 Installation of Rational Asset Analyzer

Download a trial version of Rational Asset Analyzer from the following URL:

http://www.ibm.com/developerworks/downloads/r/assetanalyzer/index.html

You also need the following software products installed on your machine to set up Rational Asset Analyzer successfully:

- IBM DB2 Version 9.0 or later
- IBM WebSphere Application Server Version 6.0 or later
- Open Object REXX

You can obtain detailed instructions about the hardware and software prerequisites, installation, and configuration of Rational Asset Analyzer in the *Rational Asset Analyzer Installation Guide:*

http://publib.boulder.ibm.com/infocenter/rassan/v5r5/topic/com.ibm.raa.doc/PDF/admhins.pdf

5.3 Rational Asset Analyzer usage models and application development stakeholders

Rational Asset Analyzer is a sophisticated product with extensive function and deep coverage. But, the learning curve is not long, because three distinct sets of Rational Asset Analyzer users or stakeholders exist:

- Project managers use Rational Asset Analyzer's metrics and graphical reports to understand and address issues of code complexity, personnel assignments, and cost.
- Analysts might be lead technical developers, application analysts, system analysts, or business analysts. This role uses Rational Asset Analyzer for application-wide maintenance requirement analysis and project scoping and for uncovering functional, performance, and application behavior.
- Programmers and programmer/analysts use Rational Asset Analyzer to understand individual programs, batch jobs, transactions, and lower-level granular elements, such as a program's control or data flow.

We discuss various use cases that are associated with these three roles, but first, we look at the product so that you understand its interface and approach to capturing and presenting metadata.

5.3.1 Rational Asset Analyzer web interface

You can open the Rational Asset Analyzer home page by pointing your browser to http://<*ip address:port no*>/dmh/index.jsp.

After opening Rational Asset Analyzer, you are most likely prompted for the user id and password combination that you used to install DB2. After that, you see a menu bar, search bar, and dashboard at home or *landing page* (Figure 5-3). The Rational Asset Analyzer home page shows totals for the Rational Asset Analyzer inventory, a breakdown of the types of software language components parsed, and access to Search and Help functionality.

From the Rational Asset Analyzer landing page, you see two toolbars with options that are common on most of the Rational Asset Analyzer web interfaces. Four tabs describe the metrics on the current metadata repository. The "current metadata repository" refers to *all of the parsed information about the software assets that you have inventoried with this Rational Asset Analyzer server.* You use the list of Help functions to learn about Rational Asset Analyzer, Rational Asset Analyzer's metadata repository, and how to use Rational Asset Analyzer.

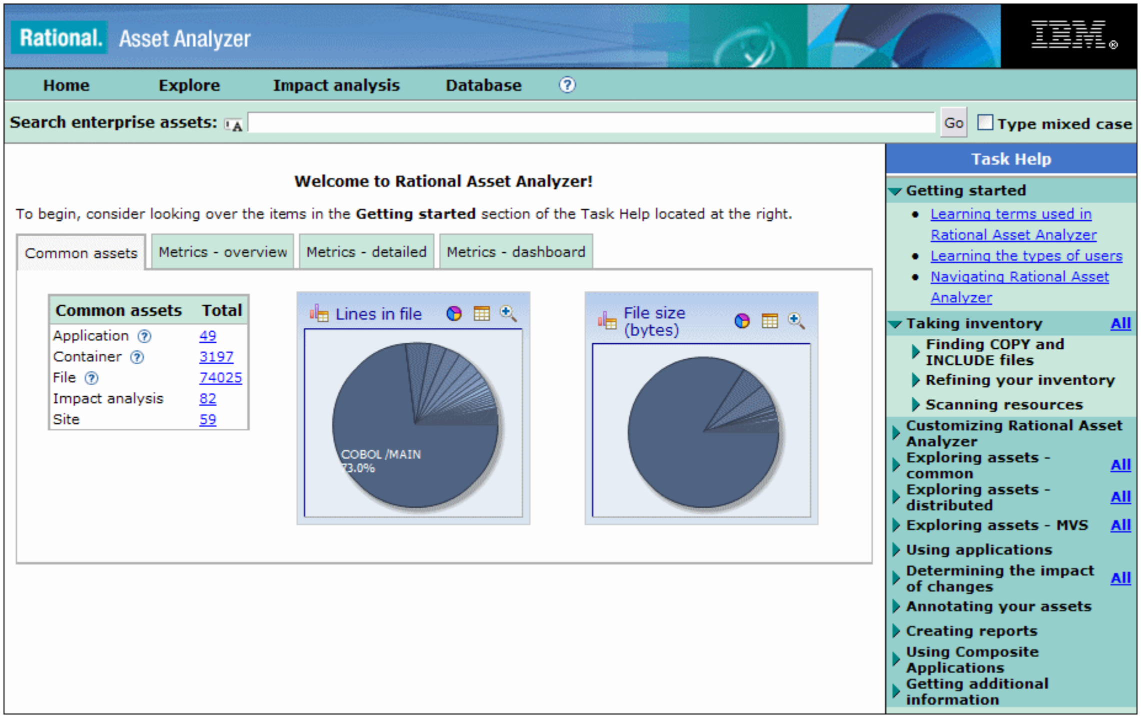


Figure 5-3 Rational Asset Analyzer home page

5.3.2 Common options

In addition to the menu options, which we describe next, every Rational Asset Analyzer web page has a "Search enterprise assets" option across the top. This function allows you to drill into the IBM Rational Developer for System z metadata repository directly.

The following menu options are available:

- Home:

  The Home option is a hyperlink back to the Rational Asset Analyzer home page dashboard.

- Explore:

  The Explore option shows a drop-down menu that provides options to help you investigate the inventory of your Rational Asset Analyzer assets.

  Each menu item displays the assets or a subset of the assets that are organized as MVS assets, distributed assets, Web Services, applications, sites, and so on:

  – MVS Assets: Displays the MVS assets inventory further categorized by asset type.

  – Distributed Assets: Displays the Java and web assets inventory categorized by asset type.

  – Web Services: Displays Web Services assets inventory by asset type.

  – Applications: Displays and allows you to manage application groups. An *application group* is a logical method of subsetting all metadata assets in the repository. You can use application groups to partition inventory assets across any logical boundary, such as by business application subset, batch versus online, and so on.

  – Containers: Displays the inventory of assets by the server folder that contains the application source files.

  – Files: Displays a list of all the *physical source files* that are in the inventory. (These files are not application data files; these files are source files.)

  – Sites: Displays the inventory by the site. A *site* is another partition that allows you to parse with various scanning options, which can be useful if you have a mixture of OSVS, COBOL II, and Enterprise COBOL programs to analyze, for example.

  – Bookmarks: Displays and manages Rational Asset Analyzer bookmarks, which are simply web pages or URL favorites.

  – Custom Queries: Displays and manages Rational Asset Analyzer custom queries. You use custom queries to extend Rational Asset Analyzer's metamodel and to answer analysis questions that are not part of the installed product.
  – User-defined relationships: Displays and manages user-defined asset relationships. This option only appears if you have administrative authority.
- Impact Analysis:

  Impact Analysis menu items enable you to display and manage Impact Analysis reports. You see two options:

  – View mine: View a summary of the analyses that you have performed.

    Note that this option implies that you have saved your Impact Analysis reports to the metadata repository and that they can be revisited and reviewed at a later time. This view provides you with the ability to compare two separate versions of an application in a "before and after" type comparison of your system.

  – View all: View a summary of all analyses that have been performed by all users.
- Database:

  Database menu options are primarily for a Rational Asset Analyzer administrator and, except for Execute Query, are only shown to a Rational Asset Analyzer administrator:

  – Collect Inventory: Initiate inventory collection (parsing and cataloging the metadata) for your z/OS application source.
  – COBOL and PL/I scanning options: This option helps in managing different inventory scanning options for COBOL and Pl/I source code.
  – Analysis queue: This queue is a real-time monitor for all of your Rational Asset Analyzer analyses. Note that because Rational Asset Analyzer is a client/server model, multiple users (clients) can run Rational Asset Analyzer analytics simultaneously. This queue shows the status of the Rational Asset Analyzer engine in satisfying the analysis requests.
  – Work queue: This option helps you to monitor the status and manage the work units in the work queue.
  – Distributed scan roots: This option initiates an inventory of Java and web application source assets.
  – Runtime environments: This option also helps you manage the inventory options for distributed inventory.
  – Java Processor patterns: This option describes various Java pattern detection features.

  - Errors: This option displays a summary of inventory scanning process errors. This key option shows the quality of the inventory, for example, missing copybooks or program source.
  - Unreferenced assets: Displays assets that are not referenced by other assets. This option is useful in answering questions, such as which copybooks are not referenced in any program.
  - Execute a query: You use this option to run a Rational Asset Analyzer custom query.
- Search Bar:

  You can use this option to search for any asset by its name (including wildcard search values). We discuss this option more later in this section.

5.4 Rational Asset Analyzer analytics and interfaces

We describe specific use cases and analysis models in this section, but in general, other than the Rational Asset Analyzer home page, the following categories of information presentation (views) or web page formats are provided in Rational Asset Analyzer:

- Metrics pages
- Explore asset pages and search facilities
- Browse, Search, and search refinement pages
- Asset Summary Lists
- Asset Detail Analytics
- Graphical Analysis Diagrams
- Source views

5.4.1 Metrics pages

In 5.6, "Rational Asset Analyzer and source metrics" on page 234, you see that, after Rational Asset Analyzer parses and scans your code, it presents a number of useful source code complexity analytical diagrams, such as pie charts and lists. These diagrams represent industry-standard views of the types of languages inventoried and the application complexity, broken out by languages, and further broken down to the level of individual programs within language categories.

5.4.2 Asset exploration pages

As shown in Figure 5-4, the asset exploration pages list MVS inventoried source files that are categorized and organized by the elements that are separate entities within the metadata repository. You can click one of these categories to drill down into the resources, for example, into batch jobs, IMS transactions, and so on. From this page, you can perform text/standard searches, you can also perform advanced searches, which allow you to select additional qualifying criteria.

Web Services and Java asset exploration pages also exist, but these pages are not shown because they are out of the scope of this book.

Technical analysts, systems analysts, and programmer/analysts use these MVS assets, and project managers might be interested in the level of detail that is provided by the asset exploration pages.

From the MVS assets page, you can perform these tasks:

- Click an asset element hyperlink, for example, CICS transactions, programs, batch jobs, variables, and so forth, which opens an Asset List (Summary) page on the selected element (Figure 5-5 on page 216).
- Search for assets, which is necessary in the example in Figure 5-4 because this repository has parsed 4,573,533 application variables. Obviously, a list interface to this number of variables is senseless and takes a lot of time to *materialize* (open). By searching for an inventoried asset, you open a subset summary list, which looks like and is functionally equivalent to the Summary type page.

  Note that the Advanced Search option allows you to scope your search request by Rational Asset Analyzer Application, site, or annotation (an *annotation* is a custom text entry).

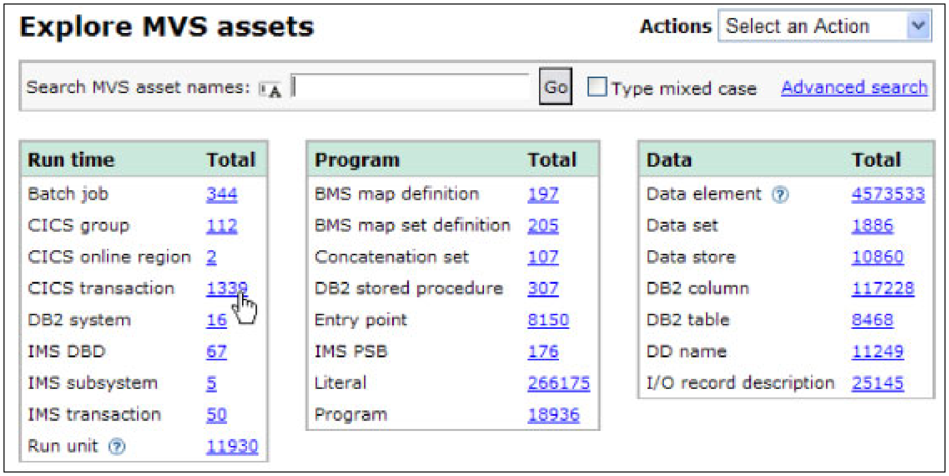


Figure 5-4 MVS asset drill-down Total column represents the total number of inventoried elements

5.4.3 Asset summary list

Discovering or learning an application or an application's elements often involves browsing or searching through selected assets, which we refer to as the *elements* of the metadata that is inventoried by Rational Asset Analyzer.

From Figure 5-5, you can see that the summary list pages organize selected elements in page-able lists that provide you with brows-able access to these functions:

- Hyperlinks to detailed analysis pages.
- Hyperlinks to additional and related elements. In the example that is shown in Figure 5-5, for any CICS transaction, you can explore the run unit for the transaction. A *run unit* is a Rational Asset Analyzer term that describes all of the operational program modules in a call/calling chain (Figure 5-6 on page 217).
- Options to annotate the CICS transaction. An annotation is a custom entry that you make to add useful technical or business information about the element, for example, this transaction is run after the batch cycles complete in order to zero out the data collection files.
- Options to delete the metadata entries from the repository.

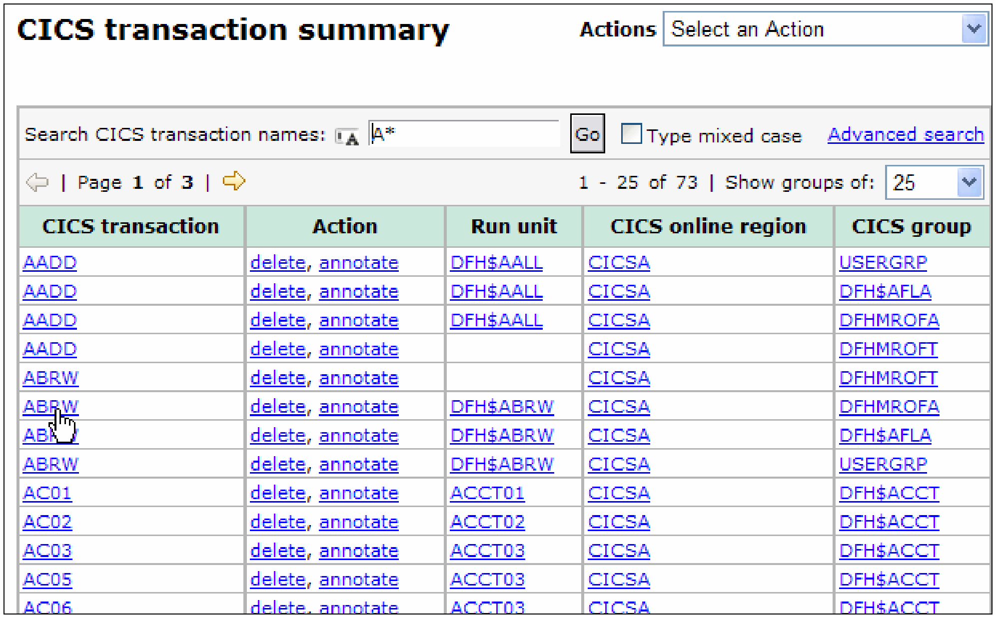


Figure 5-5 Rational Asset Analyzer summary list allows you to hyperlink to actions and resource details

5.4.4 Asset details analytic page

If you click any entry in a summary page, you open an asset details page. Figure 5-6 on page 217 shows the run unit details (metadata) for the DFSSAM04 run unit, which consists of three calls to separate program entry points, and more.

Each details analytic page presents separate metadata, depending on the type of element (asset) that is being browsed or searched. For example, Figure 5-7 on page 218 shows a detail analytic page for a COBOL program. Note these characteristics of the page:

- The Details area of the pages differ.
- The information (metadata) that is captured and displayed differs.
- The number and types of tabs, each of which shows separate relevant information about the asset, differ.

You can also see that certain information that is presented in a details analytic page is information in text format. In Figure 5-6, notice the listing of called subprograms. Figure 5-7 on page 218 shows the number of lines, comment lines, blank lines, *splitting nodes* (the number of conditional branches in the PROCEDURE DIVISION), and other informational statistics.

There are tabs for additional metadata, as well as analytics and additional hyperlinks. Additional detail information is presented as a hyperlink to additional (and related) metadata. For example, in Figure 5-7 on page 218, there are links to every COBOL copybook that is referenced in the program.

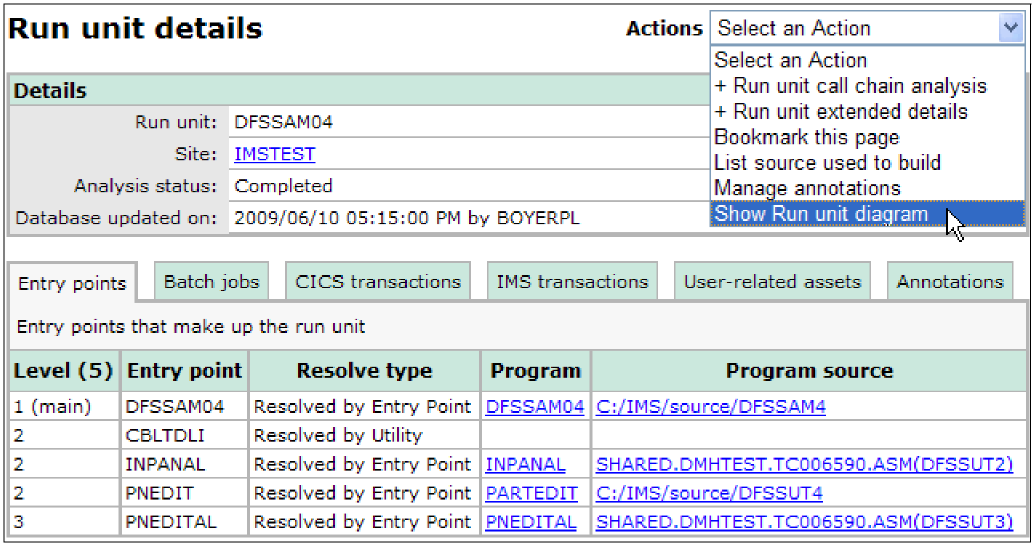


Figure 5-6 Rational Asset Analyzer Asset run unit details page

5.4.5 Actions

All of the Rational Asset Analyzer summary pages and detail pages have an Actions drop-down menu with metadata-specific element functional options. In the Run unit details Actions list in Figure 5-6, you can browse the call chain and the extended call chain, bookmark this page, list the source files that were used in building the analytics, create or update (Manage) annotations for this run unit, and generate (Show) a run unit diagram (which is shown in Figure 5-8 on page 220).

In Figure 5-6, in the Actions list options, the plus signs (+) to the left of the first two options in the Actions list denote that those actions have been saved as custom queries, which is a Rational Asset Analyzer feature.

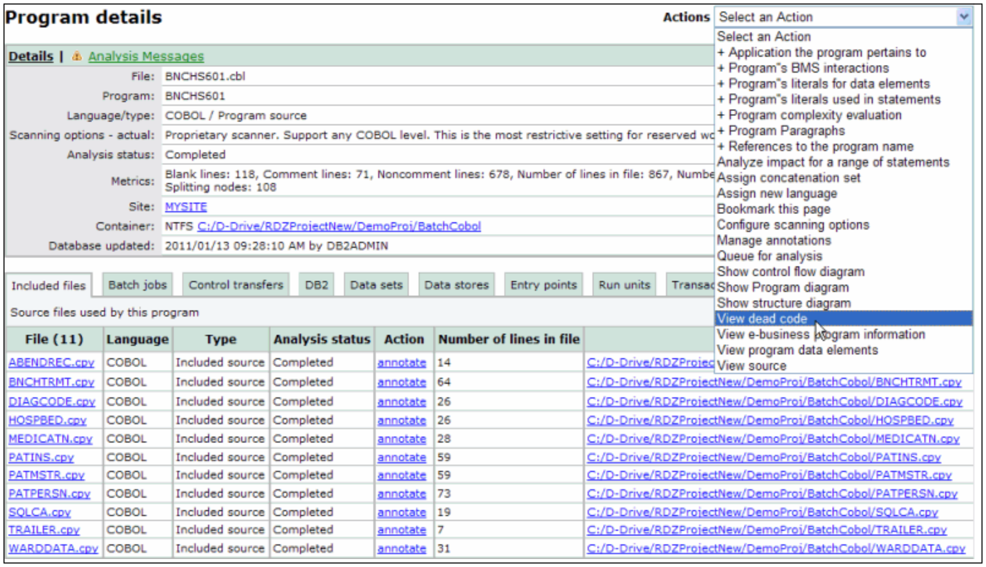


Figure 5-7 Program details page

5.4.6 Graphical analytic diagrams

Certain metadata is conveniently organized, learned, and viewed as lists or related elements, which allows you to browse, search, learn, and answer analytical questions quickly by understanding these aspects:

- Elements
- Attributes of elements
- Relationships among elements

But, certain types of analysis are accelerated through the presentation of the information in a graphical or visual context. For example, in Figure 5-8, you can see that several transactions (ADDINV, ADDPART, and DLETIN) call or invoke the run unit DFSSAM04, which in turn invokes CBLTLDI, INPANAL, and PNEDIT.

You can see this relationship in the Detailed Summary list page. But, what is not shown on that page is that PNEDITAL is invoked from PNEDIT. So, the two-dimensional graphic is semantically richer than the one-dimensional list in Figure 5-7, because the list does not show this meaningful information.

Many graphical analysis pages are available, including the following types:

- The run unit diagram (Figure 5-8 on page 220) shows a group of programs starting at the entry point called from a batch job or, in this case, a CICS transaction. This diagram shows the transactions that access DFSSAM04 and its callable interfaces, such as programs, DL/I databases, and the 3270 interface.
- The batch job diagram (Figure 5-9 on page 220) shows all of the steps, data definition (DD) statements, programs, files, and databases and shows how the data sets are passed among the steps in the job.
- The program diagram (Figure 5-10 on page 221) shows at a glance all of the job steps, load modules (programs), DD statements, data files, and data set usage throughout the job. Figure 5-10 on page 221 shows a batch DB2 program and all of the file usage (input, output, and both as indicated by the arrow directions), as well as called programs.
- The program control flow diagram (Figure 5-11 on page 221) shows all of the paragraphs or the structure of the control flow within a program.

Note that all of the hyperlinks that are shown in these diagrams are true hyperlinks that open a Detail Asset page about whatever element you click. The paragraphs in the program control flow diagram position the source code at a given paragraph (see the integration with Rational Developer for System z in 4.7, “Integration with Rational Asset Analyzer” on page 160 for more information).

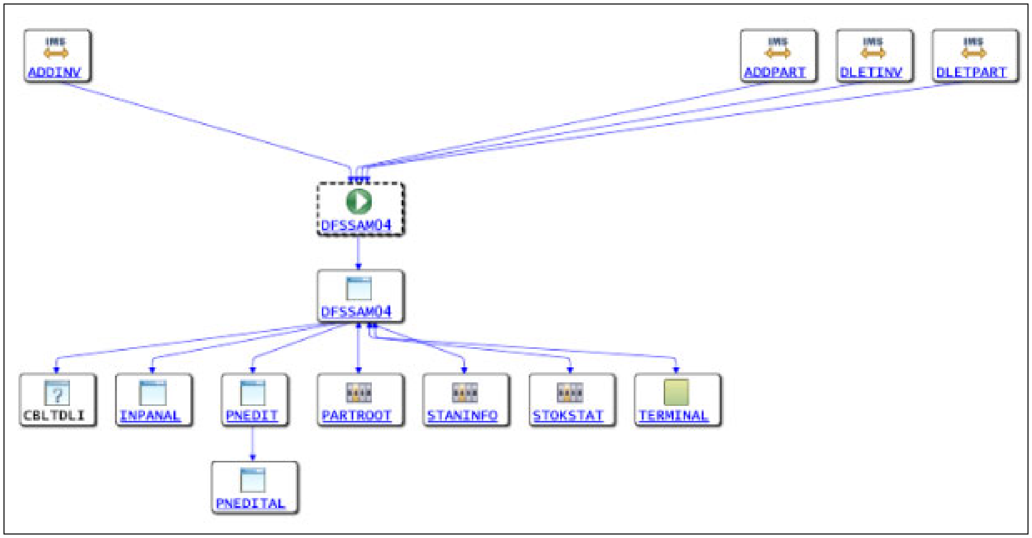


Figure 5-8 Run unit diagram

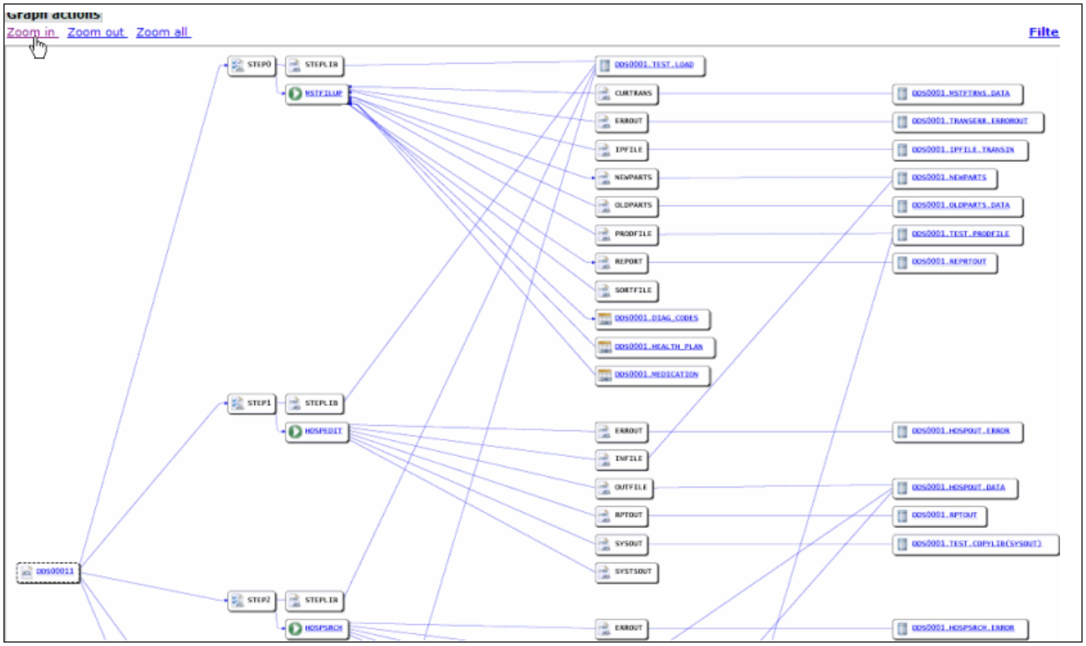


Figure 5-9 Rational Asset Analyzer batch job

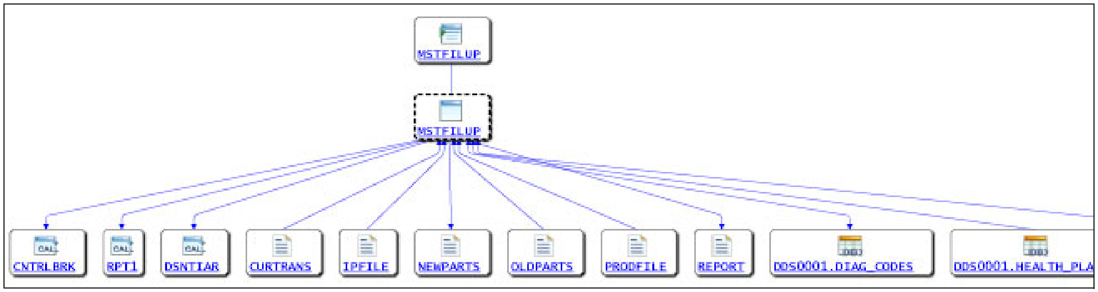


Figure 5-10 Program diagram showing a batch DB2 program and all file usage (input, output, and both as indicated by the arrow directions)

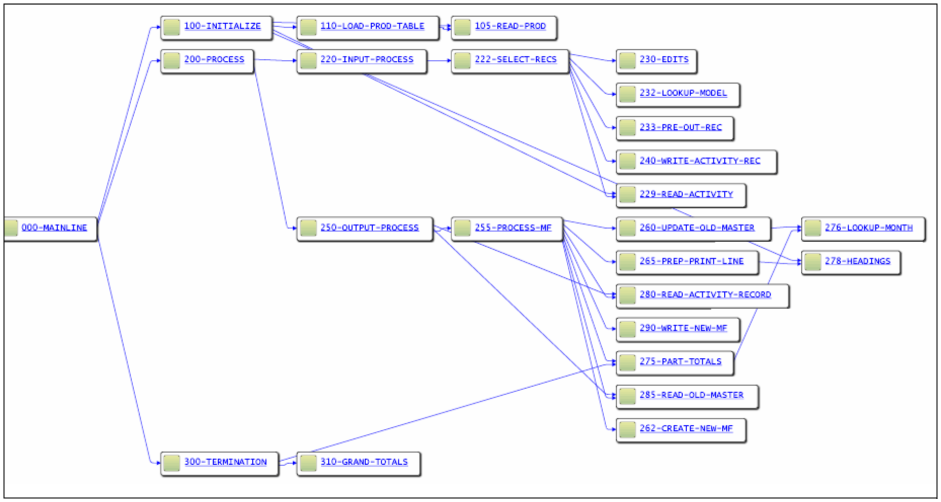


Figure 5-11 Program control flow diagram shows the paragraph and control flow structure

5.4.7 Source views

No static code analysis tool is complete without the ability for you to go directly to the source of the metadata in your application. Rational Asset Analyzer has a Source view, which is available from the Actions drop-down menu in the Details pages of most but not all metadata elements. Not all elements have source associated with them, because certain types of elements, such as the run unit, are inferred.

Figure 5-12 shows a sample source view for a data element that is part of the TRMNT program. Because it is part of the program's WORKING-STORAGE (and not inside a copybook), the View Source option displays the entire program with the following useful semantic information:

- The originating source file from the inventory scan (source location).
- A hyperlink to the Rational Asset Analyzer metadata on the program (TRMTMNT). Clicking this hyperlink displays the Program details page (Figure 5-7 on page 218).
- The declaration (data definition line).
- All lines where the variable is referenced but where the variable's contents are not modified by COBOL operations, for example, the "from" portion of a MOVE operation, a comparison, the UNTIL part of a Perform, and so forth.
- All lines where the contents of the variable are modified by a statement, for example, the receiving portion of a MOVE operation, an arithmetic calculation, VARYING, and so forth.
- The intact source code.

Note that you can either hide the line numbers, as shown by the option in Figure 5-12, or you can show the line numbers.

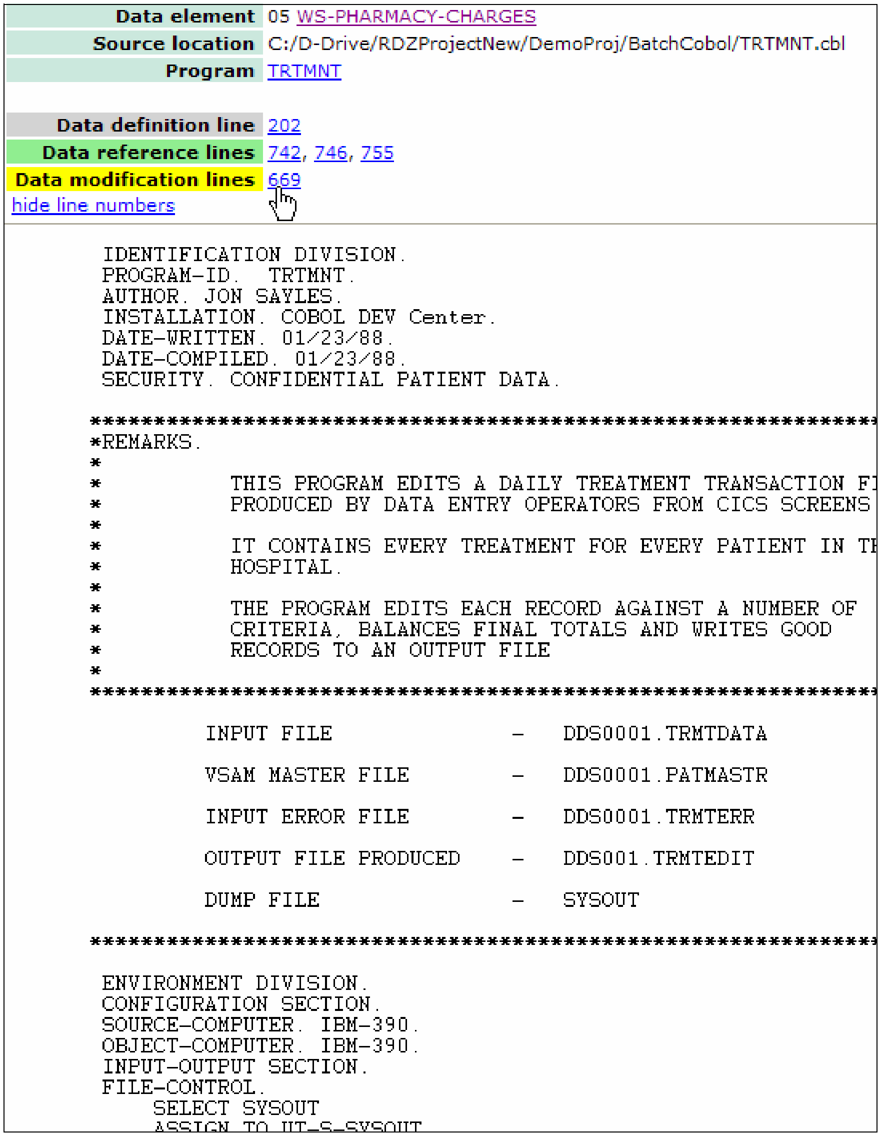


Figure 5-12 Rational Asset Analyzer source view of a search for a variable within a program

5.5 Scanning source artifacts with Rational Asset Analyzer

In this section, we discuss scanning MVS assets with the help of the Rational Asset Analyzer graphical user interface (GUI).

Follow these steps:

1. Start by launching Rational Asset Analyzer. From Microsoft Windows, select **Start** → **Programs** → **IBM** → **Rational Asset Analyzer**, or point your browser to

   `http://<ip address:port no>/dmh/index.jsp`

2. When Rational Asset Analyzer asks for your DB2 credentials (user name and password):

   a. Either get these credentials from your database administrator (DBA).

   b. Or, if you use a local Rational Asset Analyzer server, enter the local IBM DB2 Universal Database™ (UDB) user name and password that you specified when you installed the DB2 product.

3. From the Database menu on the Rational Asset Analyzer home page, select **Collect inventory**, as shown in Figure 5-13.

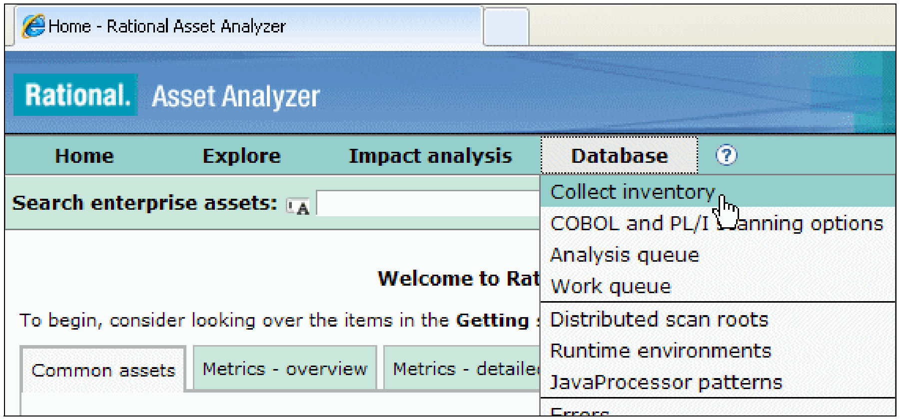


Figure 5-13 Collect Inventory: Initial step from Database menu

The window that is shown in Figure 5-14 opens.

As of the software release that we used to write this book, all of the application source code to be inventoried by Rational Asset Analyzer had to be on a Microsoft Windows machine (desktop or server). So, prior to taking inventory, we had to copy all of the source code to a local Microsoft Windows folder. We will specify that Rational Asset Analyzer scan the inventory from that folder.

Scanning source code from the local folder is an eight-step process.

4. Specify a site for all assets (Figure 5-14).

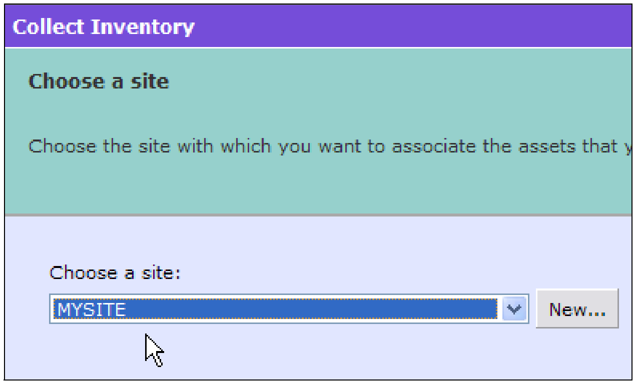


Figure 5-14 Specify a site for all assets

We create a new site and give it a name. We name it MYSITE, as shown in Figure 5-14. Click **Next.**

5. Specify the complete path for the Microsoft Windows folder to be scanned for source assets, as shown in Figure 5-15. We also select the check box for **Scan specified directory and all subdirectories**. All subdirectories of the specified folder will be scanned. In case you want to restrict the source scan to a specific type of file name, you can specify it in the Limit file search text box. The asterisk (*) is the default in this text box, which means that all files are selected. Click **Next**.

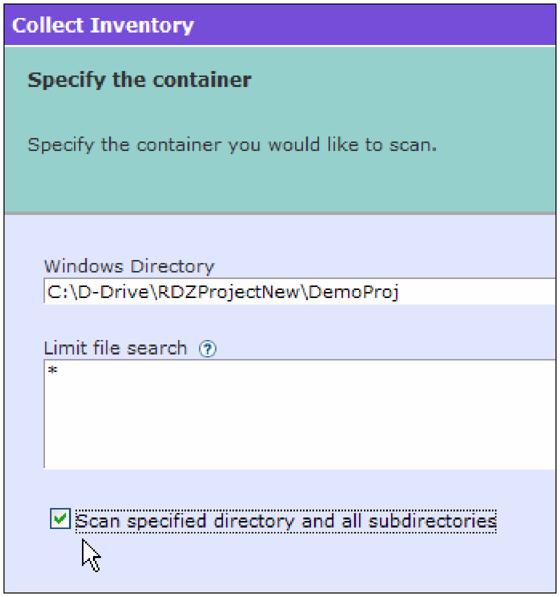


Figure 5-15 Specify the path to the folder containing source assets and scan options

6. In next two steps, specify separate COBOL and PL/I scanning options, as shown in Figure 5-16. The Rational Asset Analyzer proprietary scanner is an internal language scanner that has been provided since Rational Asset Analyzer Version 1.

   The Rational Asset Analyzer compiler-based scanners offer more robust scanning capabilities by using the Enterprise compilers. They have been available since Rational Asset Analyzer Version 4. The compiler-based scanners are available in two variations: regular and enhanced COPYBOOK/INCLUDE support.

   We select compiler options based on the type of assets being scanned. Click **Next**.

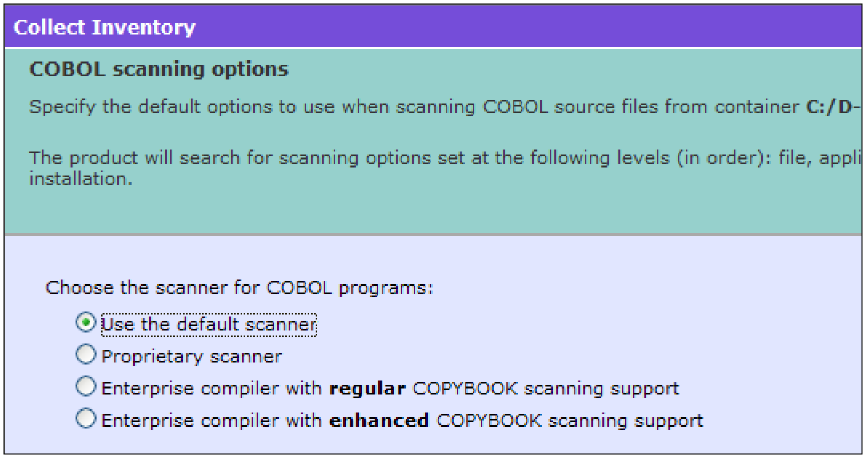


Figure 5-16 COBOL scanning options

7. Assign all of the assets that are being scanned to an existing application or to a new application. In our case, we have created a new application called HOSPITAL_MAINTENANCE for our assets, as shown in Figure 5-17. Rational Asset Analyzer applications allow you to subset a large enterprise code base and to limit searching, analyzing, and reporting to a portion of the code base that is specific to your requirements. Click **Next**.

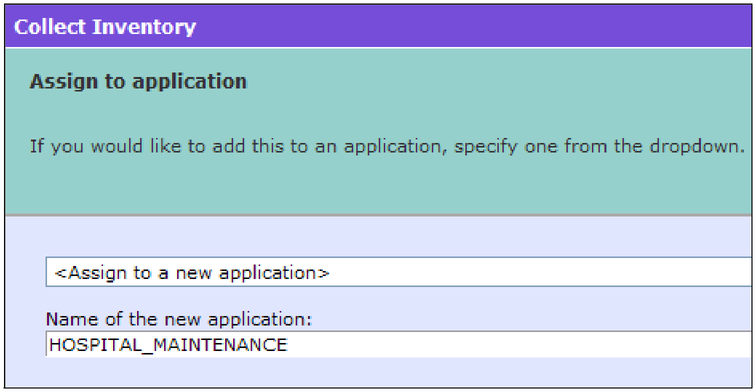


Figure 5-17 Assign scanning assets to an application

8. Assign concatenation sets for the assets, as shown in Figure 5-18.
   A *concatenation set* is an ordered list of containers, such as additional file directories and subdirectories, to be searched to resolve references to the included source, for example, JCL PROCs, assembler macros, or COBOL copybooks. We assign the concatenation set and click **Next**.

   You can also create new concatenation sets here to define alternate folders to search to resolve source dependencies.

   Rational Asset Analyzer applications and sites are, in essence, containers that you can use effectively to partition the entire aggregation of source code to inventory into subsets for more effective use. For example, you might inventory an entire division's (or even enterprise's) source code, but you want to provide a project-level analysis, as in a business project-level analysis, to a subset of the enterprise code.

   You can also use Rational Asset Analyzer applications to scope the extent and depth of a detailed Impact Analysis. We describe impact analyses in 5.11, "Impact analysis" on page 278.

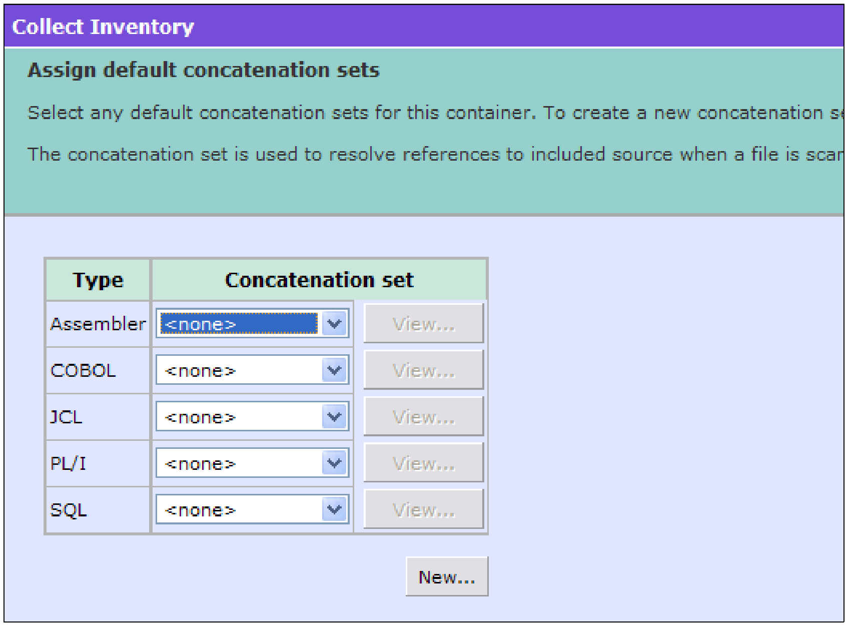
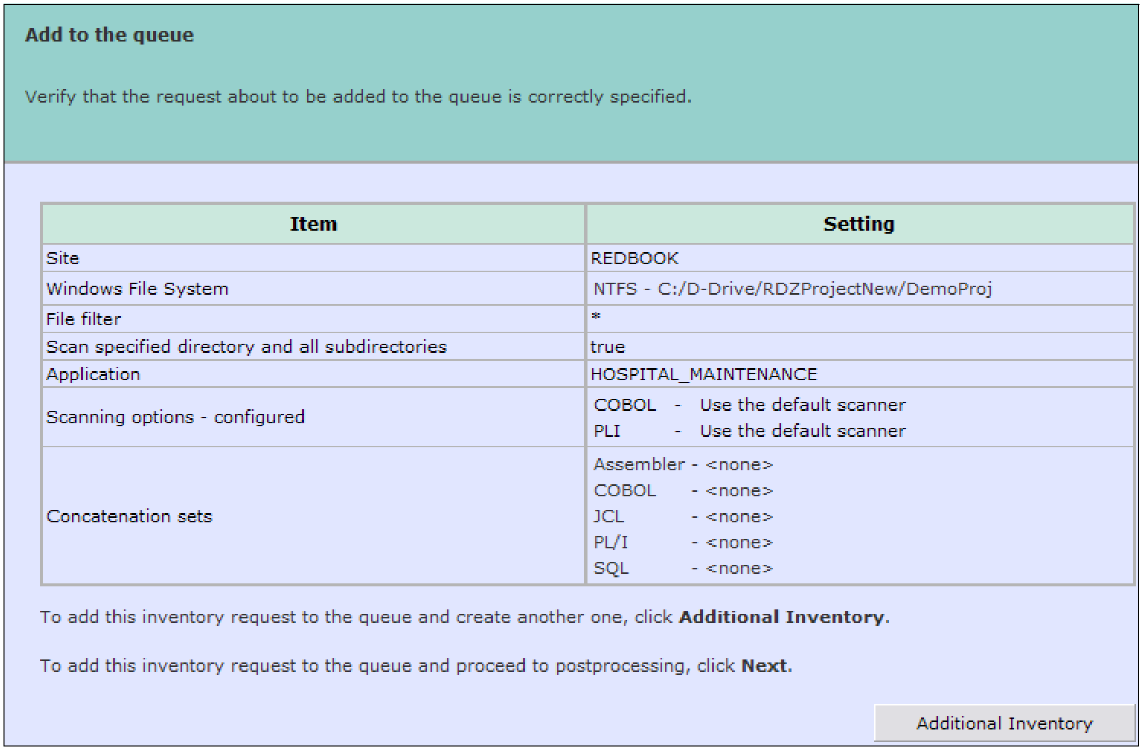


Figure 5-18 Assign concatenation sets for scanning assets

9. When a summary page displays all of the choices prior to starting the inventory parsing (Figure 5-19), click **Next**.



Item	Setting
Site	REDBOOK
Windows File System	NTFS - C:/D-Drive/RDZProjectNew/DemoProj
File filter	*
Scan specified directory and all subdirectories	true
Application	HOSPITAL_MAINTENANCE
Scanning options - configured	COBOL - Use the default scanner PLI - Use the default scanner
Concatenation sets	Assembler - <none> COBOL - <none> JCL - <none> PL/I - <none> SQL - <none>

Figure 5-19 Summary of inventory to collect and parse

10. After providing all of the information about assets (as we have done here), submit the request for scanning in the analysis queue (Figure 5-20).

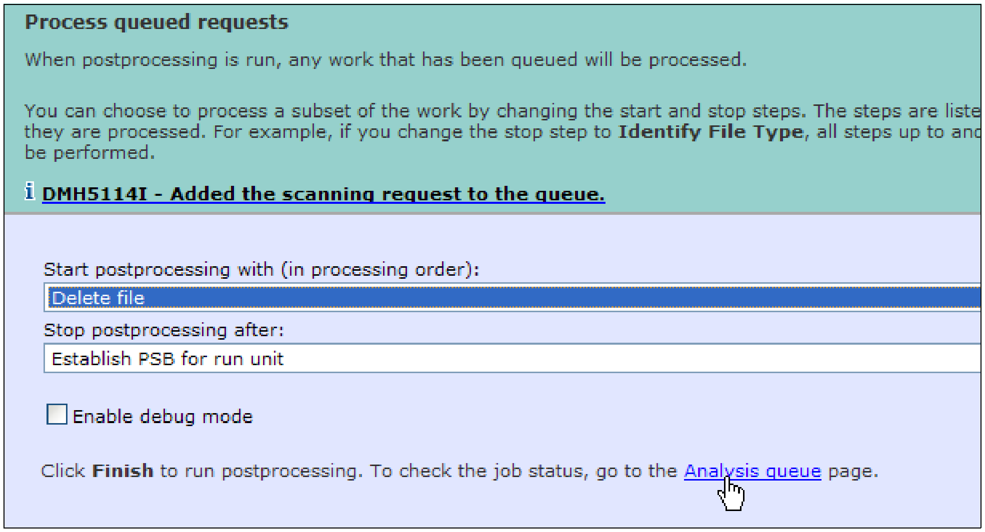


Figure 5-20 Finish inventory collection and parsing

The analysis queue follows a specific order to process asset information. We can change our request to be restricted to specific steps instead of running the whole process by selecting values from the "Start postprocessing with (in processing order)" and "Stop postprocessing after" drop-down lists (Figure 5-20). For example, if we change the stop postprocessing step to "Identify file type," all steps from the starting step up to and including that step will be performed.

The following list shows all of the inventory collection steps in the order in which they are processed:

1. Delete file:

   This step specifies that the metadata of a particular file must be deleted. For each queued work item, the metadata of a file is deleted from the repository.

2. Analyze container:

   This step initiates a container scan to gather the list of files in the container and record them in the repository. Each file name gathered automatically queues a request to identify the file language for the file or files recorded during the container scan.

3. Identify file language:

   This step identifies file contents by initiating a classification scan. The language of the file is recorded in the repository. Depending on the language code, such as COBOL, PL/I, or JCL, that is detected, a request to identify the file type might be automatically queued for the file.

   Rational Asset Analyzer does not depend on a file naming convention, file mapping, or file extension to identify the language.

4. Identify file type:

   This step identifies the file type, such as MAIN, INCLUDE, PROC, or MAP, by analyzing metadata about the file that has gathered up to this point. If the file is determined to be a main program or batch job, a request to the analyze file step is automatically queued.

5. Analyze file:

   This step analyzes the contents of a file by initiating an analysis scan. Depending on the metadata discovered, a request to the resolve dynamic references step might automatically be queued. If any additional included files are discovered during the analysis scan, those files are entered into the queue as though an analyze container request had been run against them.

6. Resolve dynamic references:

   This step processes the data element names in control transfers (CALL, LINK, or XCTL statements) and SEND/RECEIVE MAP statements, relinks SQL references, and calculates metrics for a given file that has been successfully analyzed.

7. Build run unit content:

   This step creates the CALL hierarchy for a given run unit, as specified in a TRANSACTION, batch job, or main program entry point.

8. Rebuild run unit content:

   This step rebuilds the CALL hierarchy for a given run unit, because a program in its hierarchy has been reanalyzed.

9. Connect PSB to run unit:

   This step connects an IMS program specification block (PSB) to a Rational Asset Analyzer run unit. It happens, because metadata about both the PSB and run unit was recently entered in the repository through transaction or batch job scans. We define a Rational Asset Analyzer run unit in an upcoming subsection on terms and concepts.

10. Establish PSB to run unit:

This step establishes a connection between an IMS PSB and a run unit that cannot be automatically resolved, because there are more than one correct connection that can be made. At this point, the correct connection must be asserted by eliminating duplicate choices from the repository. Note that in our steps, we kept all of the default options.

After you click **Finish** (Figure 5-20 on page 231), an analysis request is submitted to the Analysis Queue. Because Rational Asset Analyzer is a multiuser server-based product, multiple project teams might submit inventory collection requests at or near the same time. The Analysis Queue performs a function similar to the batch job or Job Entry Subsystem (JES) queue on z/OS and shows the analysis process requests and status.

After the successful completion of the analysis request, Rational Asset Analyzer has all of the metadata loaded into DB2 tables, and users can browse the metadata from the Rational Asset Analyzer browser-based user interface (Figure 5-21 on page 233).

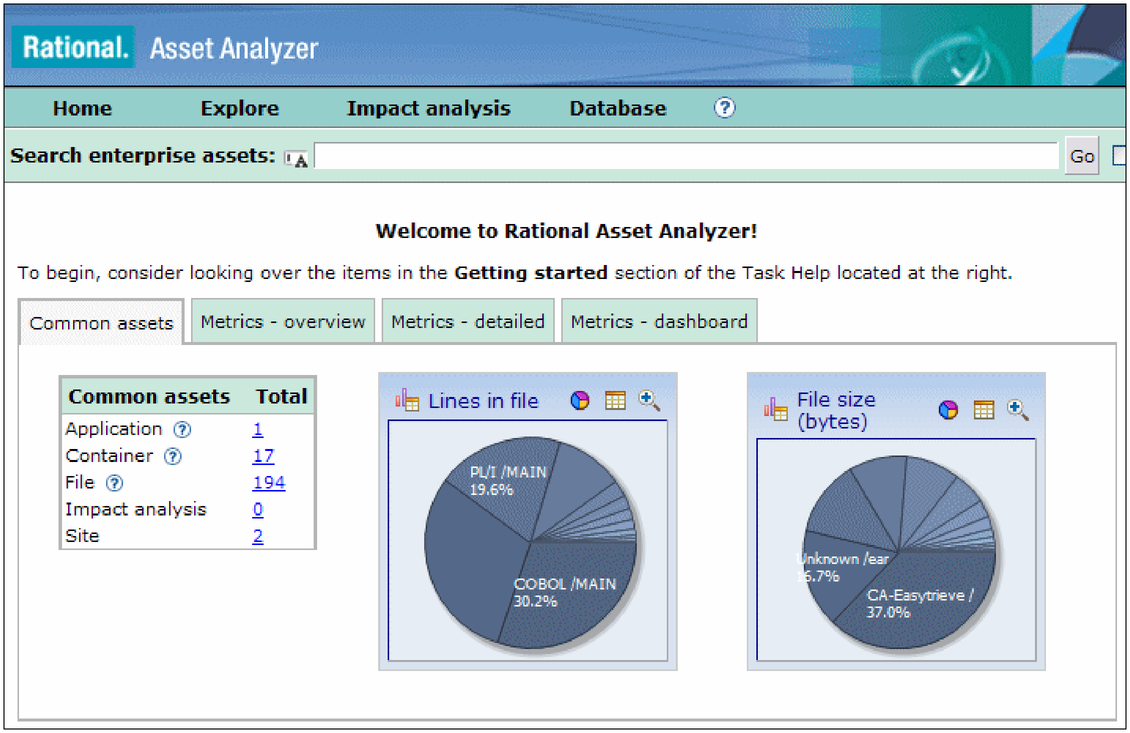


Figure 5-21 Rational Asset Analyzer home page after inventory collection finishes for our reference application

5.6 Rational Asset Analyzer and source metrics

After Rational Asset Analyzer has scanned the source code, it loads the metadata about all source artifacts into the repository or database (DB2). On completion, Rational Asset Analyzer returns you to the home page for subsequent analysis work.

Figure 5-21 shows the common assets, which are listed on the left side of the page and show your entire collected inventory. Notice the two pie charts that present a breakdown of all of the various file types that were inventoried.

For example, Figure 5-22 illustrates mousing over a pie chart, showing that 2.1% of the scanned code is either a basic mapping support (BMS) map or a Message Format Service (MFS) map. You can move the mouse pointer over the pie charts to view a high-level breakdown of the types of assets that you have scanned into your repository and the technologies and languages that underlay the assets (COBOL, PL/I, CA-Easytrieve, Assembler, and so forth). This high-level view is interesting, but the metrics that are captured during the analysis scan are of much greater use.

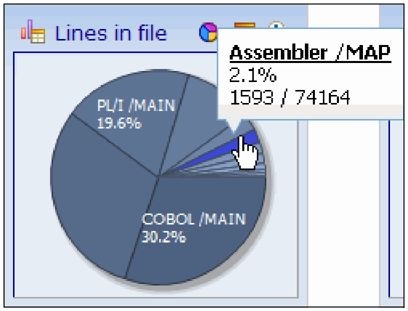


Figure 5-22 Pie chart detail shows that 2.1% of the scanned code is a BMS or MFS map

5.6.1 Metrics tabs

Rational Asset Analyzer's software metrics are used primarily in two ways:

- By project managers and supervisors, to assist in project staffing and to obtain a high-level grasp of the overall health or complexity of the code for which they are responsible
- By technical leads, project managers, and outsourcers, to understand the composition of an application and to gauge or assess an application's complexity

From the Metrics - overview tab (Figure 5-23), you can see a high-level set of complexity metrics against the entire scanned inventory:

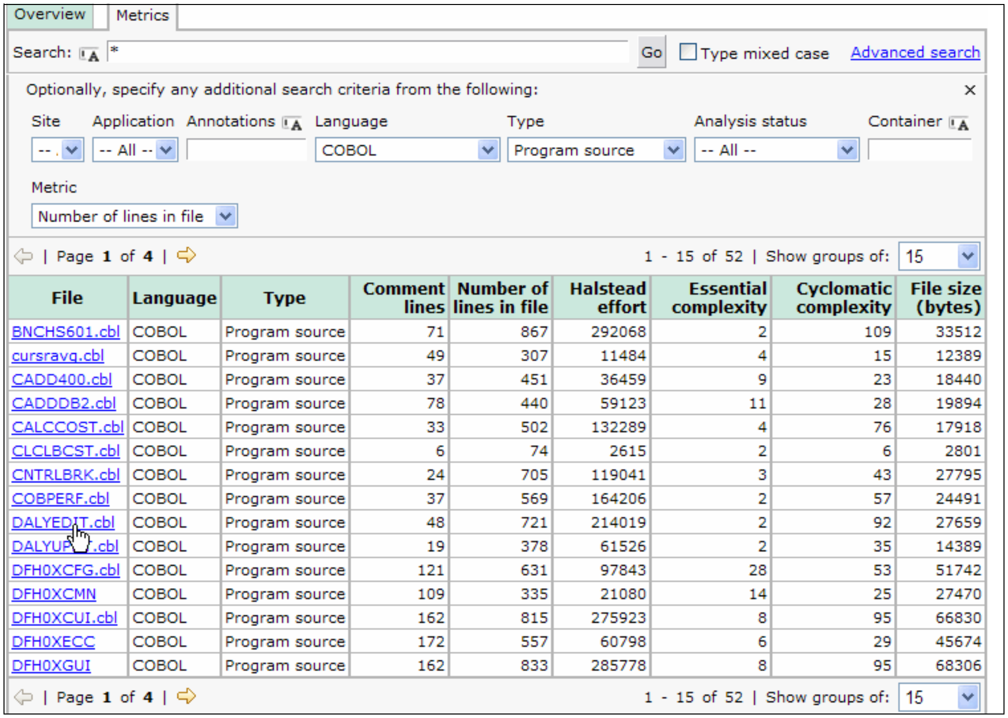
- Cyclomatic complexity: The number of independent paths through a program where an independent path is an IF statement, branch, or Perform.
- Essential complexity: A Rational Asset Analyzer-proprietary metric that determines how "structured" a program is; that is, it has no unconditional fall-through or GOTO logic that is not bound by a Perform chain exit.
- File size (in number of bytes).
- Halstead effort: The number of operands and operators found in your program. The cyclomatic complexity measures the code's *procedural* complexity, and the Halstead measures your program's *data* complexity.
- Number of lines in each source file.

Common assets | Metrics - overview | Metrics - detailed | Metrics - dashboard

Type (5)	Number of files	Minimum	Maximum	Average	Total	Standard deviation
Cyclomatic complexity	46	2	109	35	1624	30.06
Essential complexity	47	2	28	5	246	5.36
File size (bytes)	190	41	2481730	48039	9127599	220507.21
Halstead effort	51	154	1681751	121693	6206348	261089.01
Number of lines in file	187	2	15855	396	74164	1364.58

Figure 5-23 Metrics - overview tab: Complexity statistics for all of the scanned code in your repository

The scanned code is analyzed by these metrics and broken out by a hyperlink to a listing of all files (Figure 5-24), the smallest metric value found, the largest metric found, and the average metric found. A total is provided, which is useful when you want to know the total number of lines of code that were scanned into the repository and that can be associated with a project or task.



File	Language	Type	Comment lines	Number of lines in file	Halstead effort	Essential complexity	Cyclomatic complexity	File size (bytes)
BNCHS601.cbl	COBOL	Program source	71	867	292068	2	109	33512
cursravg.cbl	COBOL	Program source	49	307	11484	4	15	12389
CADD400.cbl	COBOL	Program source	37	451	36459	9	23	18440
CADDDB2.cbl	COBOL	Program source	78	440	59123	11	28	19894
CALCCOST.cbl	COBOL	Program source	33	502	132289	4	76	17918
CLCLBCST.cbl	COBOL	Program source	6	74	2615	2	6	2801
CNTRLBRK.cbl	COBOL	Program source	24	705	119041	3	43	27795
COBPERF.cbl	COBOL	Program source	37	569	164206	2	57	24491
DALYEDIT.cbl	COBOL	Program source	48	721	214019	2	92	27659
DALYUP[illegible]T.cbl	COBOL	Program source	19	378	61526	2	35	14389
DFH0XCFG.cbl	COBOL	Program source	121	631	97843	28	53	51742
DFH0XCMN	COBOL	Program source	109	335	21080	14	25	27470
DFH0XCUI.cbl	COBOL	Program source	162	815	275923	8	95	66830
DFH0XECC	COBOL	Program source	172	557	60798	6	29	45674
DFH0XGUI	COBOL	Program source	162	833	285778	8	95	68306

Figure 5-24 Hyperlinked programs and the metrics associated with each program

The Metrics - detailed tab (depicted in Figure 5-25) shows a level of detail that is between the specific program-level metrics (Figure 5-24 on page 236) and the Metrics - overview tab information (Figure 5-23 on page 235).

The Metrics - detailed tab consolidates metrics by application language. Note that you can also drill down in each of these categories as well to break the individual files out. COBOL supports the largest percentage of metrics information, followed by PL/I.

Common assets | Metrics - overview | Metrics - detailed | Metrics - dashboard

Select attribute to view: Number of lines in file

Language (12)	Type	Number of files	Minimum	Maximum	Average	Total	Standard deviation
Assembler	DBD	6	11	289	113	679	109.14
Assembler	INCL	2	12	1138	575	1150	563.00
Assembler	MAIN	8	80	260	152	1222	67.38
Assembler	MAP	9	55	473	177	1593	147.72
Assembler	MFS	4	69	145	96	387	29.24
Assembler	PSB	5	7	87	24	121	31.40
CA-Easytrieve		10	3	15855	2215	22151	4881.31
COBOL	INCL	75	2	2628	102	7673	298.64
COBOL	MAIN	48	37	1011	466	22376	273.89
JCL	MAIN	3	8	306	125	377	129.47
PL/I	INCL	3	17	1845	634	1902	856.35
PL/I	MAIN	14	31	6579	1038	14533	1851.58

Figure 5-25 Metrics - detailed tab

5.6.2 Metrics - dashboard

The Metrics - dashboard (Figure 9-26) displays a consolidated set of pie charts and tables that make up most of the metric tabs (and capability to drill down) that we have described so far in this section. Using the dashboard, you can see at a glance all of the code statistics that project managers might use to make decisions and answer questions, such as these questions:

- How complex is this application?
- Relative to other applications, will this application provide data points for scoping and costing?
- What aspects of the application are the most complex?
- What programs are the most complex? Perhaps you can assign the most complex programs to the senior staff members.

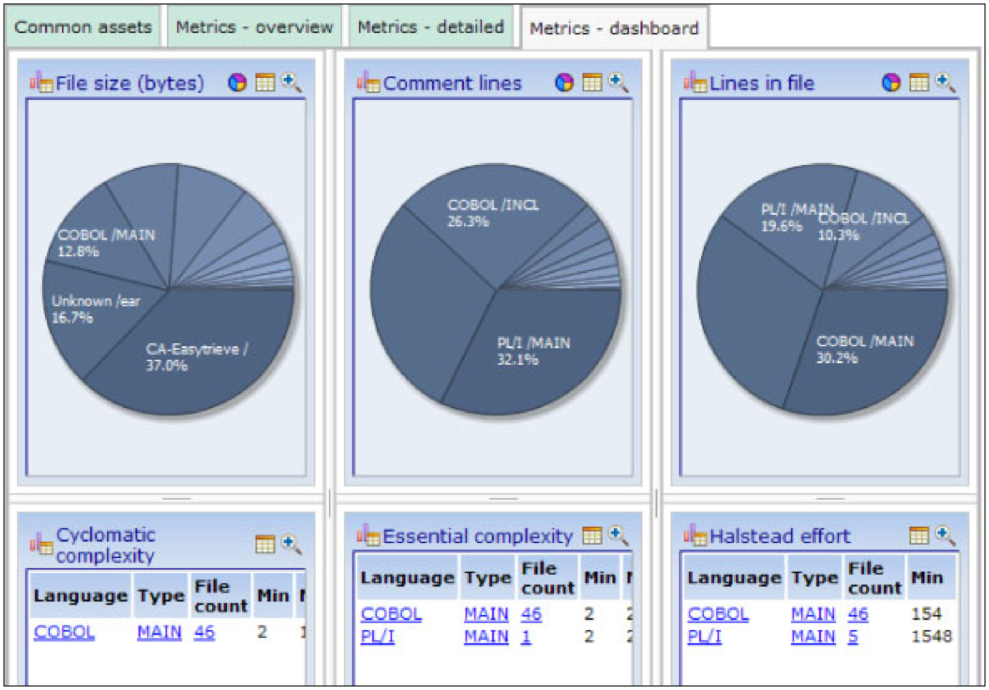


Figure 5-26 Metrics dashboard, pie charts, and lists

You can use the controls in the upper-right corner of each pie chart to toggle between a graphical view of the metrics (including moving the mouse pointer over specific details) and a tabular or list form of the metric information (Figure 5-27).

Any text that is underlined in the tables is a hyperlink that displays a search results list. The list consists of those files that match the underlined attribute or count. For example, if you click COBOL with type=MAIN, a list of all COBOL programs in the repository that have a non-zero value for the attribute, lines in file, are displayed.

If you click a link, it replaces the current page. Use the browser's back button to return. Alternatively, you can display the results in a new page if you right-click the link and select "Open in new window" or "Open in new tab".

To enlarge any particular view or table, you can click the magnifying glass with the plus symbol on it in the upper-right corner, which is the zoom in and zoom out feature.

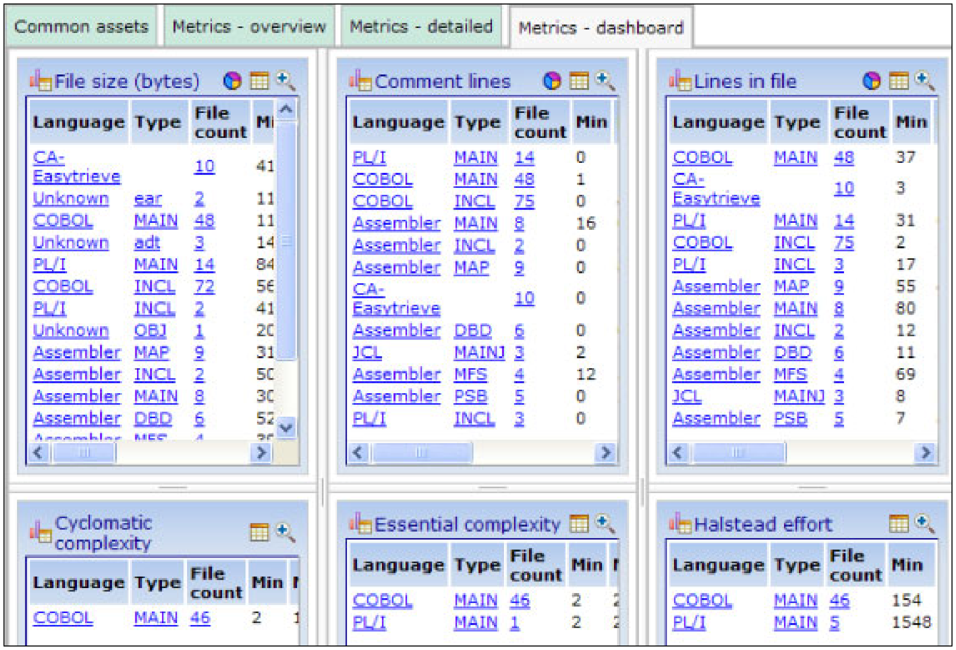


Figure 5-27 Metrics: Dashboard and all lists

5.6.3 Understanding and using metrics

You can use the application metrics to determine the composition and health of your applications. During inventory scanning, Rational Asset Analyzer calculates three industry standard metrics, cyclomatic complexity, essential complexity, and Halstead effort, and captures the file size.

Rational Asset Analyzer's software metrics quantify how difficult it is to comprehend a program and thus are a measure of its maintainability. The more difficult a program is to understand, the more it costs to maintain it. By capturing and saving this information, you can create a history of changes and begin to manage based on trends. You can customize Rational Asset Analyzer to capture and display your own metrics, as well.

The primary use of the Rational Asset Analyzer metrics is for project scoping, management tasking, and assignments:

- Use the Metrics - overview tab to gauge the overall code base's complexity through the captured statistics. You can also capture how consistently complex, large, or small the programs are through the standard deviation values (note that the lower the standard deviation, the more consistently alike the code).

As stand-alone values, these metrics are somewhat meaningful; however over time and especially when paired with your project debrief, you can use the values as relative measures of complexity in a more fine-tuned manner.

- Use the Metrics - detailed tab to break out and analyze your code base by language and run time. These values can help you to understand the following characteristics:
  - The kinds of program language technologies that are in the inventory
  - The size and complexity of the code across the technologies, which will help you staff projects accordingly
- Use the Metrics - dashboard and the program-level metrics (Figure 5-24 on page 236) to break out and analyze your code base down to the program element level. You can use this valuable information to assign maintenance tasks to senior, experienced staff versus entry-level staff appropriately, as well as handle project management scheduling and scoping at the program level.

Now, we continue our exploration of Rational Asset Analyzer by looking at its use as a tool to understand applications.

5.7 Understanding applications with Rational Asset Analyzer

Project managers and, to a limited degree, project technical leads use Rational Asset Analyzer's metrics. However, software developers, programmer/analysts, application analysts, technical project leads, and systems analysts all use Rational Asset Analyzer primarily to solve problems and understand and discover applications, architecture, and programs:

- Scope application modifications (see Chapter 12, "Trivial Change Request use case" on page 499 and Chapter 13, "Non-Trivial Change Request use case" on page 545 and specifically, the sections on trivial and non-trivial software modifications)
- Learn an application or understand the behavior of a program, batch job, or online transaction
- Determine the impact of a software change
- Trace a software defect to its root cause (see Part 4, "z/OS application maintenance and support" on page 473 and specifically, Chapter 11, "Production support use case" on page 481)

Many analytical methods and workflows are available that combine the Rational Asset Analyzer tools and interfaces that we described in 5.4, “Rational Asset Analyzer analytics and interfaces” on page 213. We start by looking at what you can do with Rational Asset Analyzer and then explain how to perform analysis.

5.7.1 Types of analysis addressed by Rational Asset Analyzer

The following list, which was originally published in *Faster Application Change and Reuse with WebSphere Studio Asset Analyzer*, REDP-4225, is extensive, but it is not a complete list of the kinds of analyses that you can perform effectively by using Rational Asset Analyzer and the kinds of questions that are answered by Rational Asset Analyzer:

- Rational Asset Analyzer examples of analytics for applications and program understanding:
  - List the batch jobs in an application
  - List the programs in an application
  - List the transactions in an application
  - View the diagram for an application
  - View the transaction flow for transactions in an application
  - See the size and complexity of your programs
  - List the batch jobs that use a program
  - List the data elements that are used by a program
  - List the data sets that are used by a program
  - List the DB2 columns that are used by a program
  - List the DB2 stored procedures that are used by a program
  - List the DB2 tables that are used by a program
  - List the programs that are called by a particular program
  - List the programs that call another program
  - List the run units that include a program
  - List the transactions that include a program
  - List the programs, transactions, and batch jobs that use an included source file
  - View the control flow for a program
  - Browse the source code for a program
  - List the batch jobs and transactions that use a run unit
  - List the entry points in a run unit
  - List the programs that make up a run unit
  - List the source that is needed to build a run unit
  - View the calling hierarchy for a run unit

- The following list shows examples of searches for data elements, data sets, and DB2:
  - List the programs that use a data element with a given name
  - List the data elements with certain data characteristics (type, length, and so forth)
  - List the batch jobs and steps that use a data set
  - List the programs that access a data set
  - List the transactions that access a data set
  - List the programs that use a DB2 column
  - List the programs that use a DB2 stored procedure
  - List the programs that use a DB2 table
  - List the columns in a DB2 subsystem
  - List the run units that use the DB2 subsystem
  - List the stored procedures in a DB2 subsystem
  - List the tables in a DB2 subsystem
  - List the views in a DB2 subsystem
  - List the views that use a DB2 column
  - List the views that use a DB2 table
- The following list shows examples of searches for IMS:
  - List the transactions, database descriptions (DBDs), program specification blocks (PSBs), online data stores, and containers (application control block (ACB), DBD, and PSB libraries) that are associated with an IMS subsystem
  - List the segments, PSBs, and data sets that are associated with a DBD
  - List the run units, PSBs, entry points, and programs that are associated with an IMS transaction
  - List the PCBs, batch jobs, and transactions that are associated with a PSB
  - List the fields in an IMS segment, as well as the run units, using a PSB that references the segment
  - List the batch jobs that use a PSB
  - List the DBDs for an IMS subsystem
  - List the IMS programs
  - List the IMS subsystems for a site
  - List the IMS subsystems that use a DBD
  - List the IMS subsystems that use a PSB
  - List the program communication blocks (PCBs) that are defined for a PSB
  - List the PSBs for an IMS subsystem
  - List the PSBs that use a DBD
  - List the segments in a DBD
  - List the transactions that use a PSB
  - View the transaction flow for an IMS transaction

- The following list shows examples of searches for CICS:
  - List the CICS files and groups for an online region
  - List the CICS programs
  - List the CICS regions for a site
  - List the files that define a BMS map or map set
  - List the maps in a BMS map set
  - List the programs that use a BMS map or map set
  - List the transactions in a CICS group
  - View the flow of transactions in a CICS group
  - View the transaction flow for a CICS transaction
- The following list shows examples of searches for batch jobs:
  - Browsing the source for a batch job
  - List the INCLUDEs and PROCs that are used by a batch job
  - List the run units invoked by a batch job
  - View a diagram of a batch job

You can use Rational Asset Analyzer to perform these tasks and many more.

5.7.2 Rational Asset Analyzer analytic search models

You can use Rational Asset Analyzer's metadata repository and web interface tools to address questions and solve problems that software developers face each day in the realm of application and program discovery, software understanding, and program analysis.

Because the scope or number of enterprise assets in a Rational Asset Analyzer metadata repository is so enormous, we typically start every analytical operation with a search to look for an item of interest that relates to our requirements.

From a high-level perspective, there are three kinds or modes of separate but related Rational Asset Analyzer search models (Figure 5-28 on page 245):

- Look at something specific. Looking at the code is how you learn software and discover what is in it and how it works. This task is often the prerequisite to the other two search models.

- Look for something specific or search for a specific item. Typically, looking for the declaration or use of a specific literal, variable, or copybook within a program. At a higher level, you can search for a specific program, file, DB2 table or database within a transaction or batch job, or BMS/MFS map within a transaction:
  - While not always the case, searching for something specific often implies or assumes that you have a fundamental understanding of what you are working with.
  - More importantly, searching for something often means you are searching for something unique (which is the exception) or specific or a general pattern:
    - Something unique that you can find with a single search text argument and that has no wildcard characters in the pattern
    - Something specific but at a high level of abstraction that requires the equivalent of an SQL LIKE search. For example, you might look for all variables that contain information about part numbers or automobile vehicle identification numbers throughout an application. Users do not universally and precisely adhere to naming standards. Therefore, you must search with one or possibly many wildcard text characters and multiple search phrases with logical ORs used by Rational Asset Analyzer.
    - Fuzzy pattern searching is the most common search method.
    - Searching for something unique is more often the exception to the rule.
- Look or search for metadata that is related to or dependent on a separate item of interest in the application. This type of search includes these examples:
  - “Where used” searches:
    - What copybooks are contained in this program?
    - What variables or programs are called from this program?
    - What windows are sent or received in a given transaction?
  - The classic impact analysis search pattern in which you look for application elements that might be affected by a change to something else in the application.

5.7.3 Browsing versus searching

Note that we use the terms *search* and *search model* in a broad context to cover standard search functionality and browsing through the assets in the metadata repository and drilling down among related or dependent assets.

We continue with examples of using Rational Asset Analyzer to accomplish the three types of analytic models. From the steps and windows shown, you can get a good idea of how Rational Asset Analyzer handles these requirements.

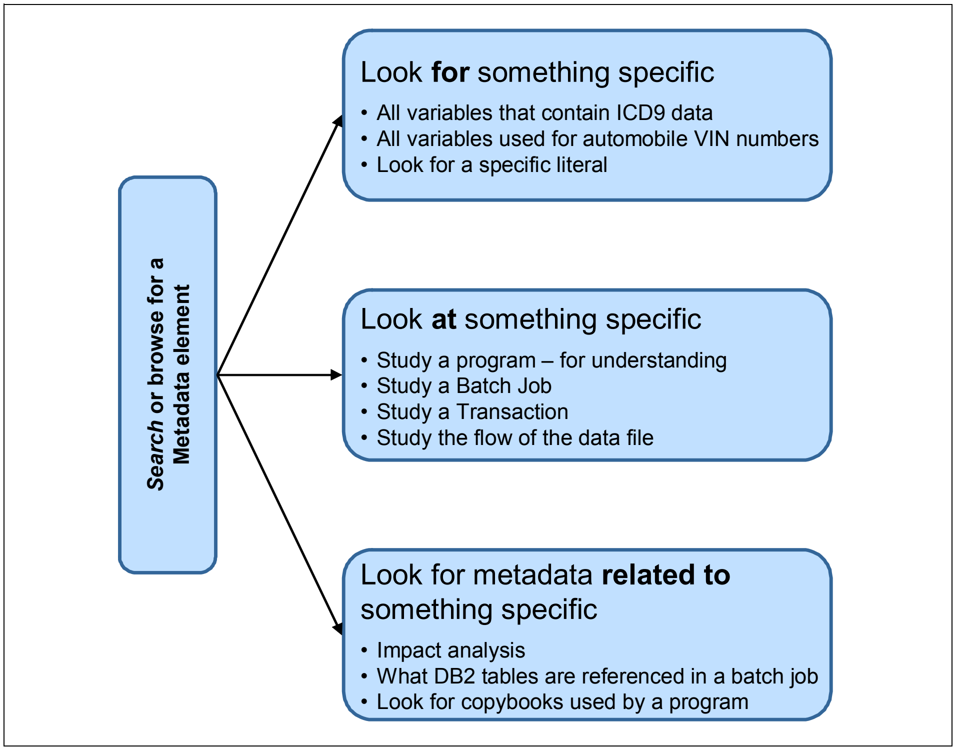


Figure 5-28 The three most common analytic search models of Rational Asset Analyzer

5.8 Analysis by looking at something specific

Performing lookup analysis using Rational Asset Analyzer and its metadata repository is absolutely the fastest and most effective approach to learning, comprehending, or discovering what is in your code.

Performing analysis by using Rational Asset Analyzer is much more effective than the more traditional code analysis alternatives:

- Reading code line by line, which, given the size and complexity of production assets, historically has proven to be a time-consuming and error-prone approach
- Reading code comments, which depends on the technical writing skills of developers and the time that they have been given to document the code
- Discussing the code with other experts and often not having access to subject matter experts in a particular analytic requirement

Next, we consider how Rational Asset Analyzer is superior. We use Rational Asset Analyzer for batch jobs, transactions, and program lookup analysis.

5.8.1 Studying or learning about a z/OS batch job

If you need to make a change to a batch job or if you support the batch job as part of a nightly batch cycle, you might need to understand the steps and follow the data set flows. You can read the JCL line by line, and if the batch job is not long and if you know z/OS JCL, this approach can work. But there is a quicker, more intuitive, and efficient way to perform this analysis by using Rational Asset Analyzer tools.

First, you need to find the batch job. You have several options:

- Search on the batch job name from anywhere under the Explore menu. Note that this search can include the wildcard text search characters:
  - Question mark (?): Meaning any one character at this byte position in the string
  - Asterisk (*): Meaning zero to any number of characters starting at this byte position in the string
  - There are many additional search options, which we describe in 5.9.1, “Searching with wildcard text characters” on page 259
- Select **Explore MVS** → **Batch job** → **Total** to open a scroll-able list of all batch jobs

- Use a combination of the previous two options. From Explore MVS, start with a search to initially filter out unwanted batch job names and then continue by clicking the remaining Total value, which opens a smaller list.

From the list, click the batch job, which opens a Batch job details page (Figure 5-29). From this page, you can see the individual job steps in sequence, and you can see information about all of the DD statements (Figure 5-30).

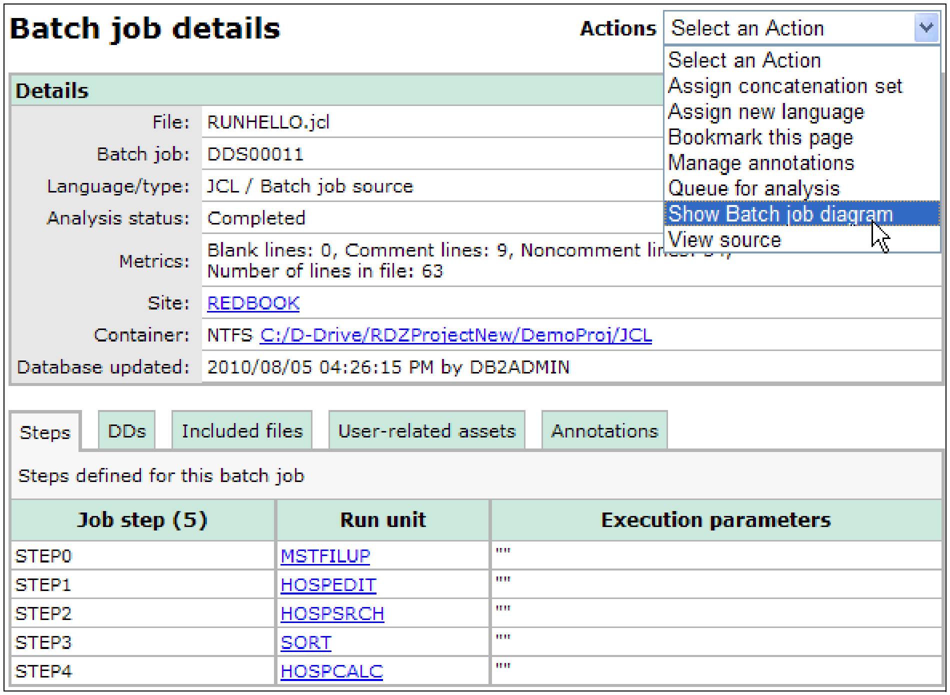


Figure 5-29 Batch job details

Figure 5-30 shows that you can click a data set name, which opens a data set detail page that shows references to the data set and the use of the data set by all of the assets in the repository. This information can be useful for understanding impact analysis from a high level, resolving file contention issues, and learning which are files are online, batch-only, or both.

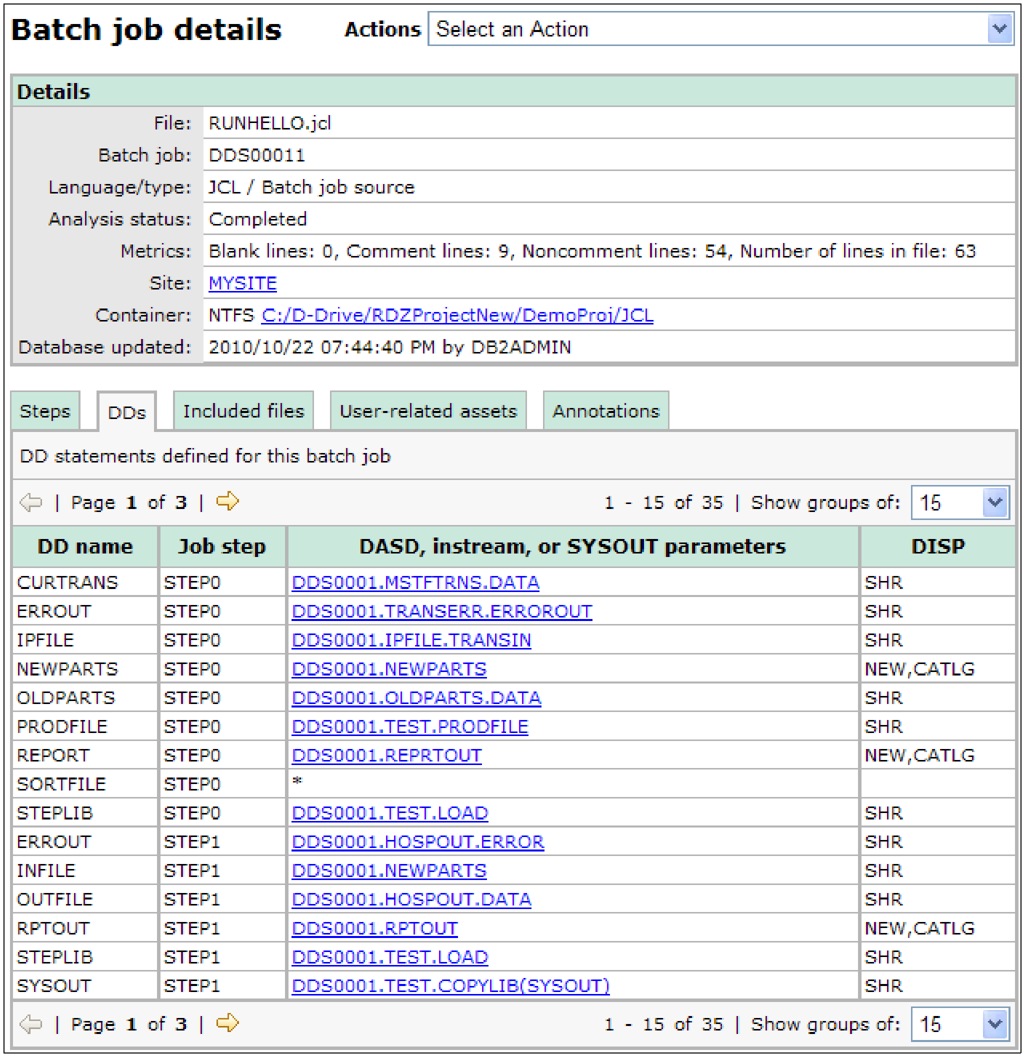


Figure 5-30 Batch job details page: DDs card tab

In Figure 5-29 on page 247, you can select **Show batch job diagram**, which opens a graphical picture of the job, as shown here in Figure 5-31. Batch job diagrams provide semantically rich graphics that can help you to clearly visualize the flow of the steps, see at a glance the data set usage throughout the batch job, and get a high-level overview of the batch job.

Batch job diagrams allow you to zoom in and zoom out for a closer look at the entire job. You can filter batch job diagrams, as shown in Figure 5-32. Filtering allows you to exclude details in which you might not be interested and to focus on one job step at a time.

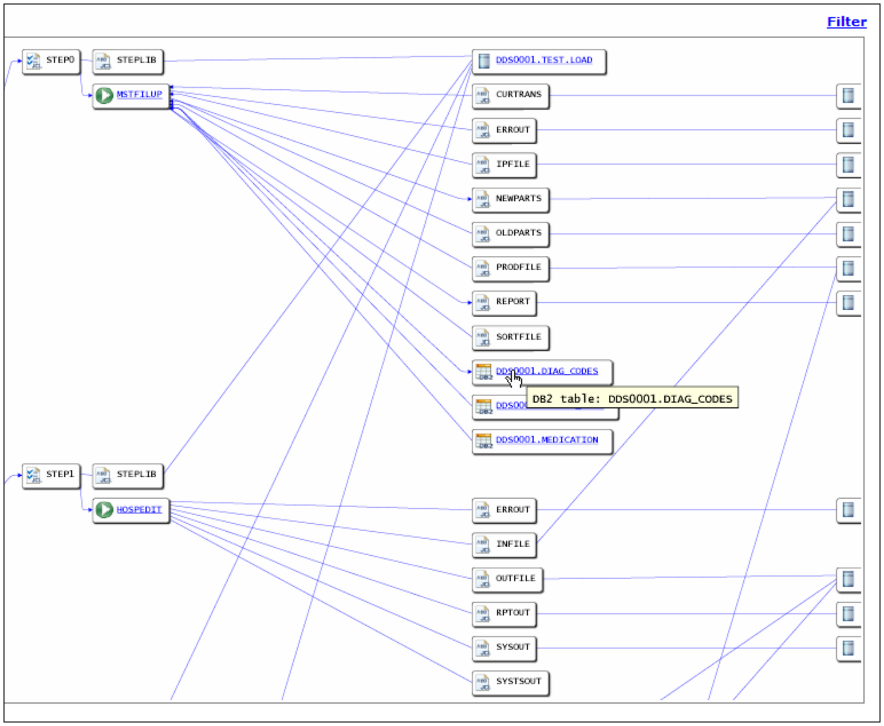


Figure 5-31 Batch job diagram

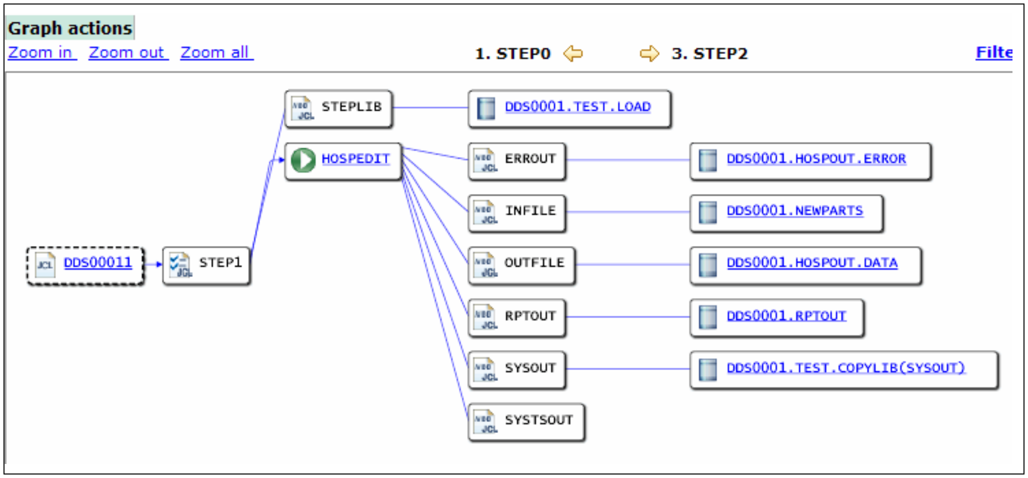


Figure 5-32 Filtered batch job diagram that shows one step at a time

Note that the filtered batch job diagram allows you to step forward and backward through the job, as well as zoom in and out.

Figure 5-32 shows the following additional semantically-rich information:

- There are several icons that represent various element types:
  - JCL icons for DD cards
  - Data set icons for libraries (including Load Libraries) and VSAM and QSAM data sets
  - DB2 tables
  - IMS database segments (see Figure 5-33 for an example)
  - The program or run unit has its own icon
- The arrows pointing to and from the DD cards are significant. They show the following information:
  - Data sets/tables/IMS segments that are *read from*
  - Data sets/tables/IMS segments that are *written to*
  - Data sets/tables/IMS segments that are both read from and written to

5.8.2 Studying or learning about a transaction

You can apply the same discovery process to most of the artifact types in the metadata repository (for z/OS applications, see Figure 5-4 on page 215 for a list). Not all of the repository artifacts have diagram tools, but all of the repository artifacts have list and detail Rational Asset Analyzer pages with valuable information about the selected artifact and its dependencies. Other helpful diagrams that you can use are the transaction diagrams and the run unit diagrams, which show online transactions and batch programs.

To get to a transaction or run unit diagram, you need to search for and filter the assets in your repository by going from the list to the detail pages:

- Search on the transaction or run unit name, from anywhere under the Explore menu.
- Go to Explore MVS and click the program or run unit category totals, which will open a scroll-able list of all entries. In a small system, this approach might work, but for production-scale assets with thousands or tens of thousands of inventoried assets, this approach is untenable. Therefore, you will need to search or combine a search with Explore MVS.

After you find the transaction (Figure 5-33 and Figure 5-34) or the run unit (Figure 5-35) detail pages, you can study the metadata by either reading the text entries, hyperlinking to a dependent artifact (Figure 5-33 through Figure 5-36 show dependent programs as linked URLs), or study the diagrams.

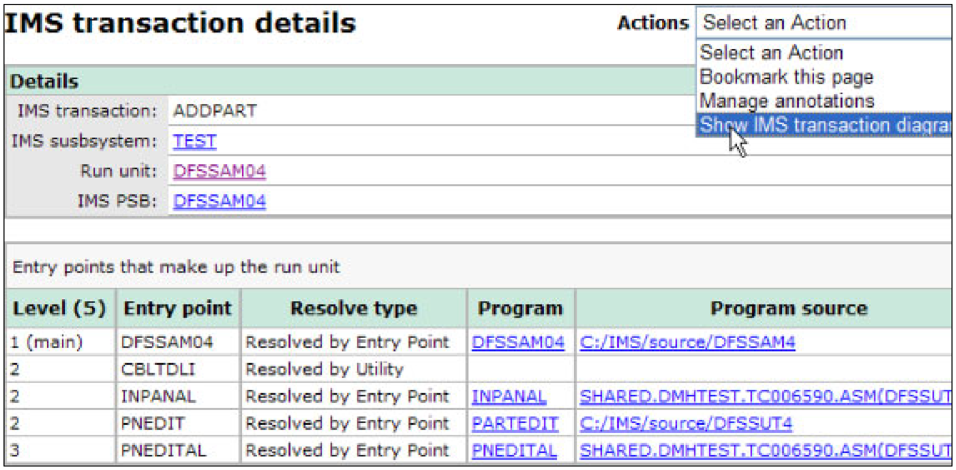


Figure 5-33 IMS transaction details page

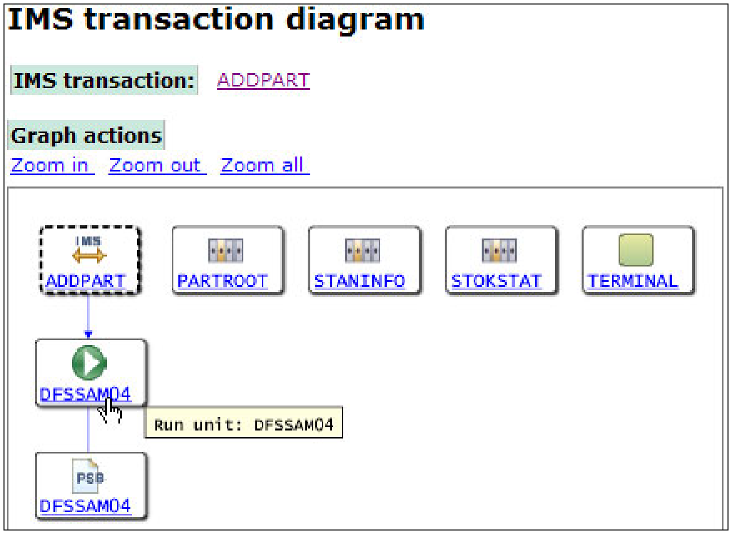


Figure 5-34 IMS transaction diagram shows run unit, database, and PSB information

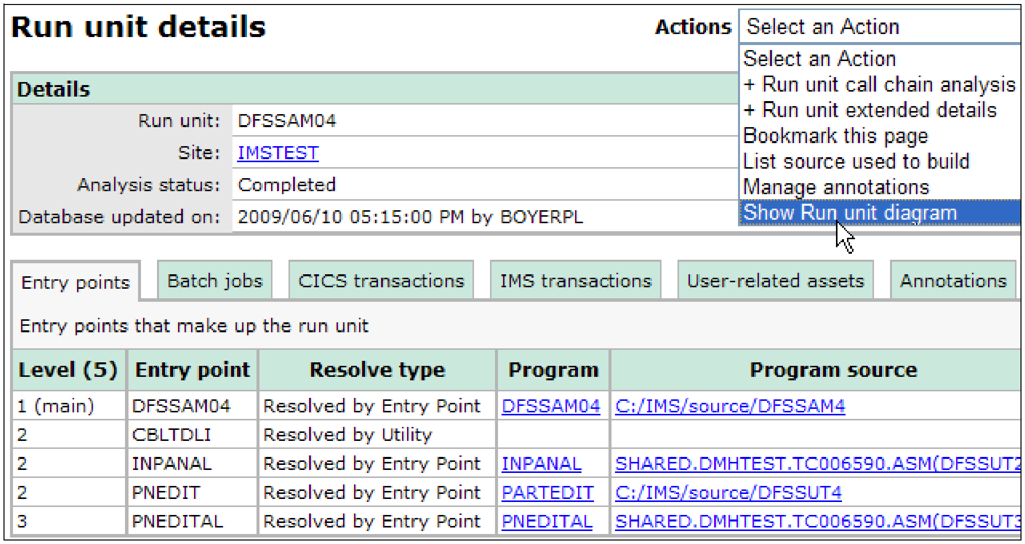


Figure 5-35 Run unit details

Figure 5-36 shows an IMS online program DFSSAM04, which is called by multiple transactions (ADDINV and ADDPART), and performs these functions:

- Reads from and writes to the message queue (terminal icon)
- Calls CBLTDLI (IMS runtime interface module for COBOL) and two other custom programs, one of which calls a third subprogram.

  The question mark (?) in the CBLTDLI icon indicates that Rational Asset Analyzer was unable to find the source code for this module during the inventory process (because this component is a shipped IBM system software component)
- Reads and writes the following IMS database segments:
  - PARTROOT
  - STANINFO
  - STOKSTAT

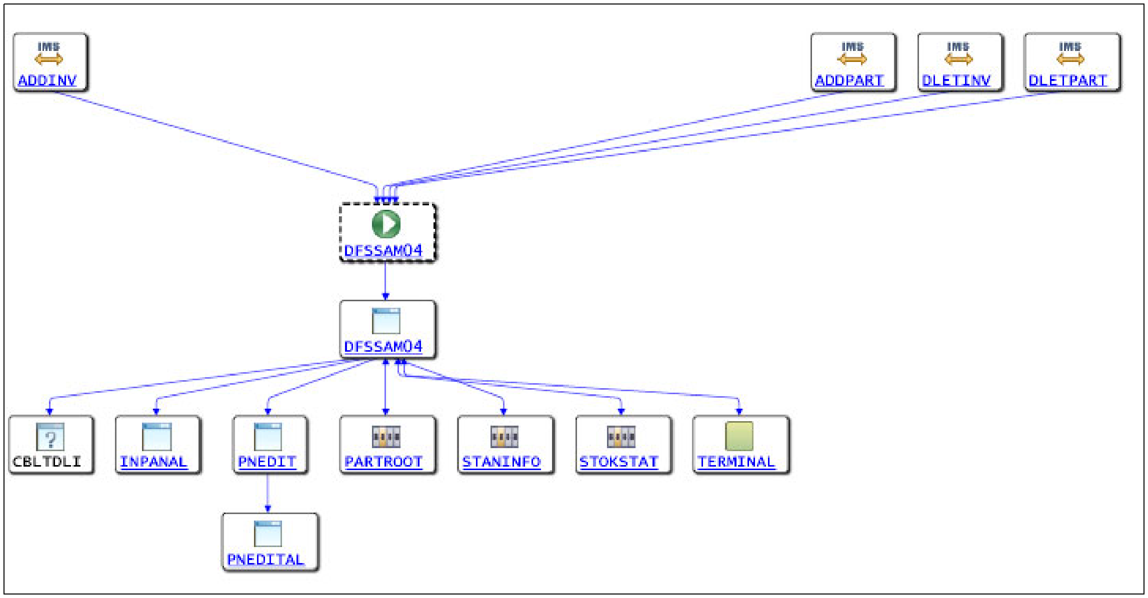


Figure 5-36 IMS Transaction Manager (TM) Data Communication (DC) transaction shown through a run unit diagram

Figure 5-37 shows a CICS transaction that is shown in a run unit diagram.

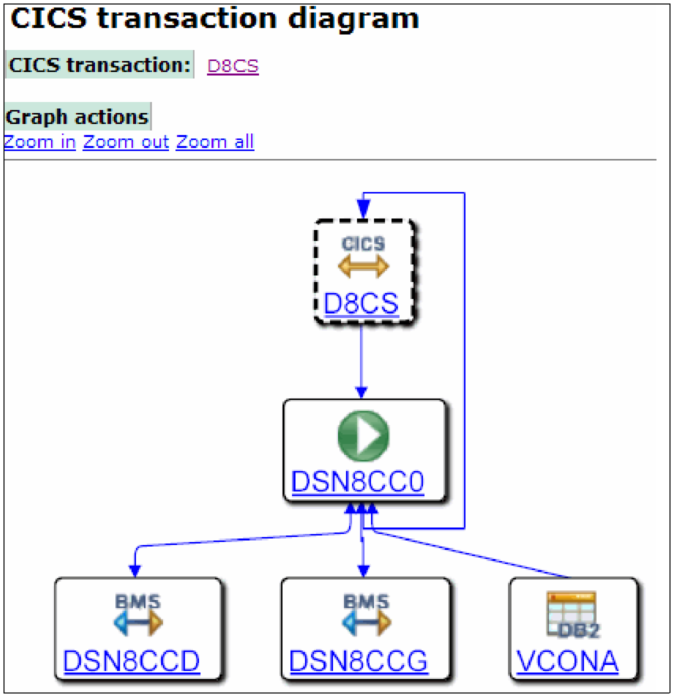


Figure 5-37 CICS transaction shown through a run unit diagram

5.8.3 Studying or learning about a program

In 4.7, “Integration with Rational Asset Analyzer” on page 160, we describe Rational Asset Analyzer integration with IBM Rational Developer for System z. This method is the optimal way to study and learn about a program. You combine the analytic tools of Rational Asset Analyzer with the analytic tools of IBM Rational Developer for System z with the edit, compile, and debug capabilities of IBM Rational Developer for System z.

However, the same kinds of diagramming and logic and data flow discovery features are present in stand-alone Rational Asset Analyzer.

Begin your workflow by searching through Explore MVS assets until you drill down (or search out) the program’s details page.

In Figure 5-38, you can see the extensive metadata that is captured and available (through hyperlinks) in the Rational Asset Analyzer repository and details pages:

- Overall information about the program language, size, number of blank lines, comments, and operational (non-comment) lines
- A vast number of referenced relationships are available in the tabs, including hyperlinks to the details pages for these assets:
  - All include or COPY files
  - All transactions and batch jobs that invoke the program, through a program's run unit (not only or not necessarily directly)
  - All control transfers (Figure 5-39 on page 257), including the parameter variable names that are sent or received to the program and from the program (to a subprogram)
  - DB2 objects, such as tables, columns, and stored procedures
  - All QSAM and VSAM data sets
  - All IMS segments and a list of the IMS calling parameters (Figure 5-38 on page 256)
  - All program entry points (they will show the program name by default, as well as ENTRY *xxx* USING statements)
  - All run units in which the program participates (recall that a run unit is a composite load module that is executed either in batch or online by a transaction)
  - Transactions
  - User-related and annotations metadata, which is custom (and usually business-specific) information that you enter, such as: "`This program runs differently at month-end`"

First, aggregate and harvest all of the rich information about a program in the graphics and Rational Asset Analyzer details pages, including the available hyperlinks to the dependent assets, and use the pull-down menu to access the Rational Asset Analyzer program diagrams. Then, you can discover or learn about a program (its behavior, complexity, and logical structure) quickly. This approach is certainly quicker than reading source code.

The more complex the program is (either because of twisted logic with lots of GO TOs or simply enormous program size), the more benefit you reap from Rational Asset Analyzer as a static analysis tool.

Let us look at a real-life example. When one of this book's authors, who had written the COBOL programs in the reference application, was documenting Rational Asset Analyzer, he noticed that, according to the program control flow diagram (Figure 5-40 on page 258), there was a fall-through problem with one of his routines. He went back to the code and read through several of the paragraphs and was unable to find anything wrong. Perplexed because the code had been tested, he was about to send a question to the Rational Asset Analyzer development team when he decided to visually check every line.

He found a GO TO the-wrong-EXIT statement that caused the fall-through and that Rational Asset Analyzer had caught and exposed clearly through the control flow diagram.

How much is it worth to discover mistakes? How much is it worth to catch a potential production ABEND from occurring? Many developers believe `Risk = Probability * Consequences`. Perhaps the probability of this particular fall-through condition occurring was small (it was not caught by its initial test data), but the consequences of production ABENDs in job rerun costs, the labor to fix them, and then performing the root cause analysis, are significant.

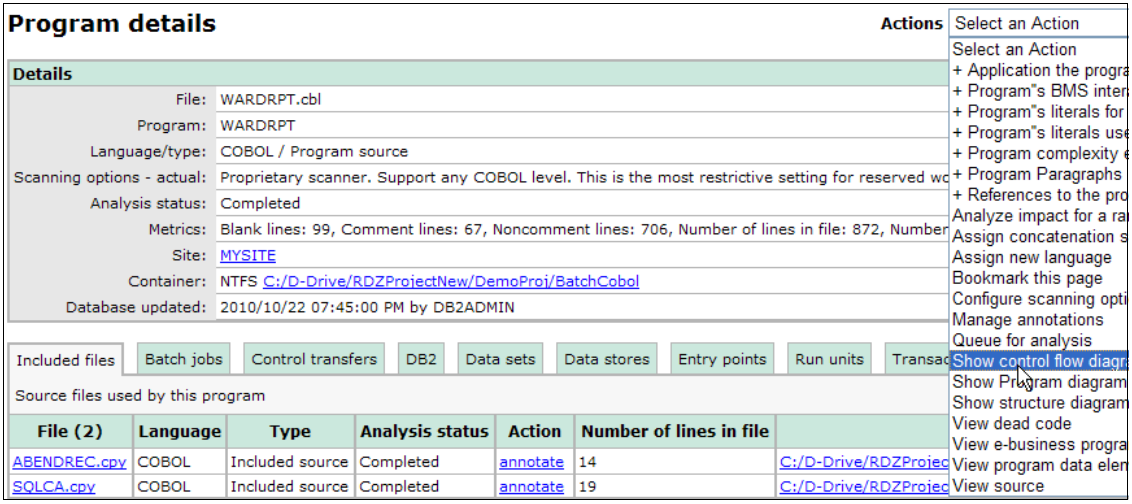


Figure 5-38 Program details page

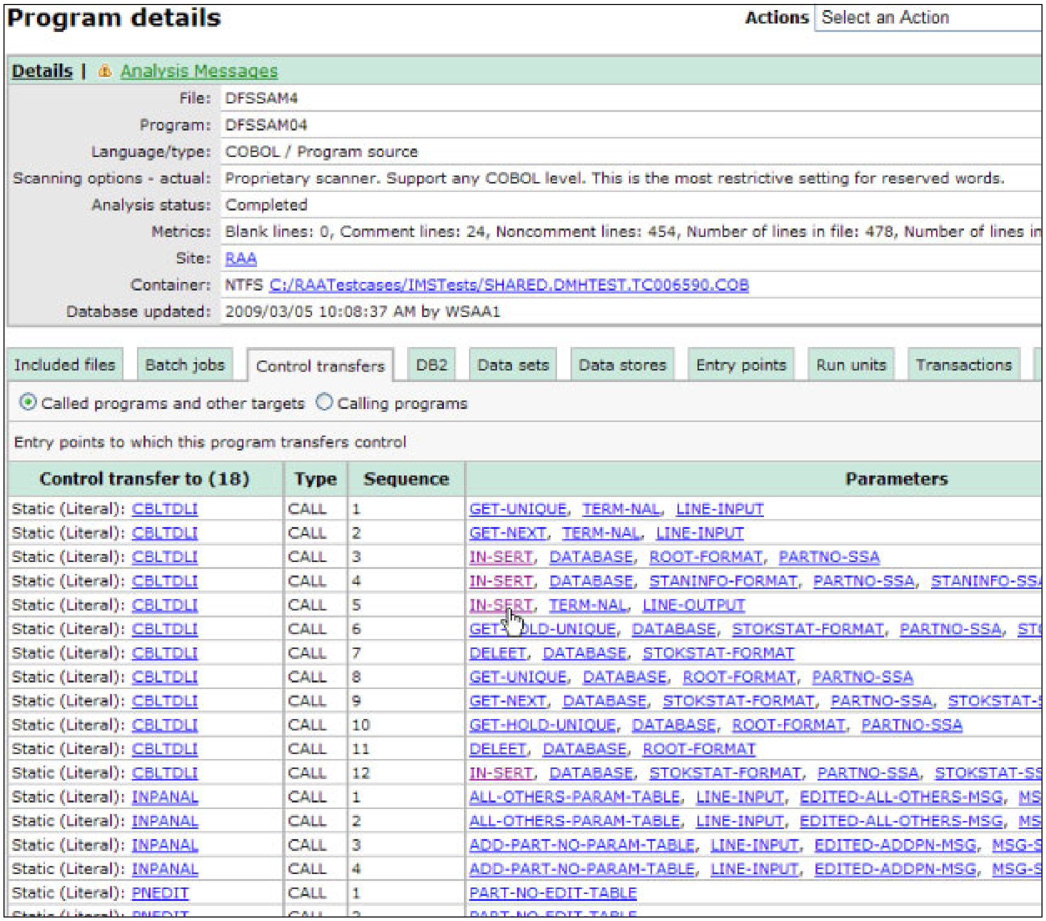


Figure 5-39 Control transfer details showing IMS DC and DL/I calls: Each call/parameter is a hyperlinked details page

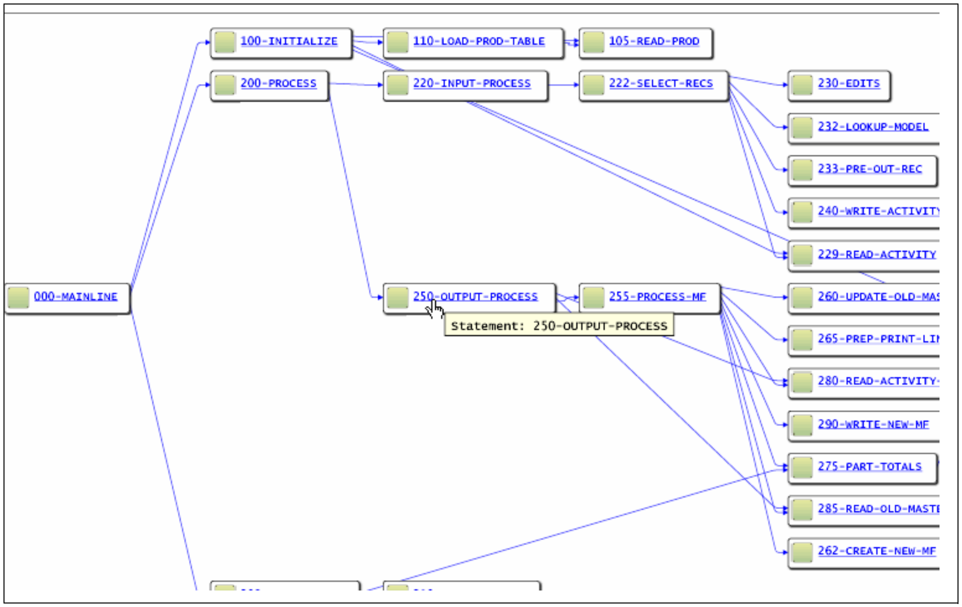


Figure 5-40 Program control flow diagram showing using the mouse on individual paragraph names. In Rational Asset Analyzer integration, the hyperlinks synchronize with IBM Rational Developer for System z's LPEX code editor

One final diagram is shown in Figure 5-41. The Program diagram is a high-level snapshot of a single program, its file, database, and table I/O, calls to subroutines, and also (if IMS or CICS) icons for online resources such as messages queues, temp storage, and so on.

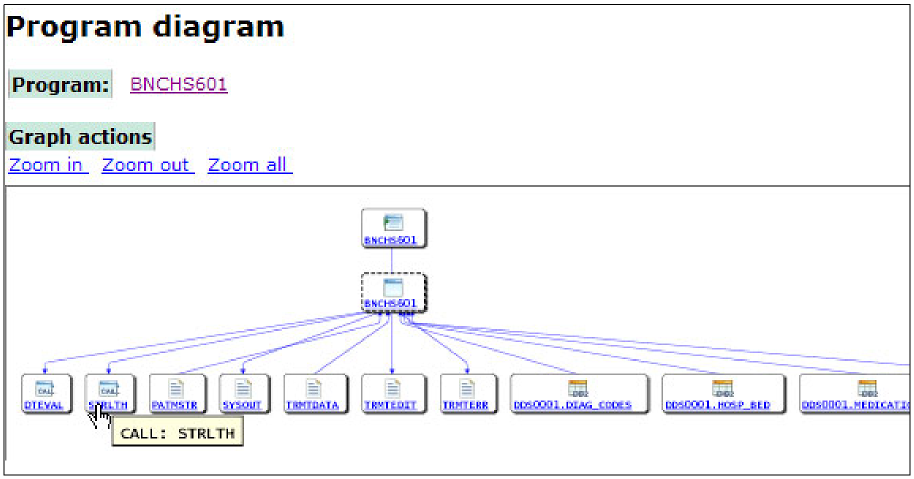


Figure 5-41 Graphical program diagram showing direct calls and file and DB2 access out of a program

5.9 Analysis by searching for something

You might think that using Rational Asset Analyzer to search for something across the inventoried assets is simple. And, it is simple. However, you are searching for an asset's identifier in the metadata model, not merely performing a text search. This method will return fewer "false positives" or results that match a semblance of your search criteria but do not help you find what you are looking for. In fact, they complicate your work, because you need to inspect and eliminate search "hits".

Because the size, which is measured in the number of artifacts inventoried for your enterprise assets, is so enormous, the ability to perform pinpoint searching is critical. Rational Asset Analyzer can perform pinpoint searching, because it has broad, deep, and common requirement-specific searching capabilities:

- Rational Asset Analyzer has a rich but not overly complex set of search parameters and wildcard characters.
- Under Advanced Search, there are specific common requirement results and additional search features, such as looking for variables with a physical or logical length within a range or searching for unused variables.

We look at Rational Asset Analyzer's search features and then describe common search patterns for z/OS maintenance work.

5.9.1 Searching with wildcard text characters

These examples demonstrate the rules that are used for common name searches. Square brackets ([]) are used to isolate the actual strings and are not part of the strings or phrases themselves:

- An asterisk (*) wildcard matches zero or more characters.

  Example: [A*] matches [A] or [AB] or [ABC].

- A question mark (?) wildcard matches exactly one character and ensures that the match is of the same length.

  Examples:

  [A?] matches [AB] but not [A].
  [A??] matches [ABC] but not [A] or [AB].

- Single-quotation marks or double-quotation marks are required for strings that have embedded blanks.

  Examples:

  ["A TEST"] matches the phrase [A TEST].
  ["ATEST"] matches the string [ATEST].

5.9.2 Searching with logical AND, OR, and NOT

Rational Asset Analyzer supports logical AND, OR, and NOT searches.

Logical OR

Multiple terms that are *separated by white space* and *not* enclosed within quotation marks are processed as a logical OR expression, for example:

- [ONE TWO THREE] matches [ONE] or [TWO] or [THREE].
- ["TEST 1" 'TEST 2'] matches [TEST 1] or [TEST 2].
- ["'SOME STRING'"] matches ['SOME STRING'].

Logical NOT

A minus (-) sign is a logical NOT operator and excludes terms. Logical NOT is not an intuitive construct for developers, because developers are taught to phrase conditional logic as positive evaluations. Often, when searching, you look for a specific pattern that is close to but separate from many results that end up as false positives. The logical NOT is handy as a false positive filter, for example, [A* -*C] matches [AB] and [ABCD] but not [ABC].

Logical AND

A plus (+) sign is a logical AND operator. The remaining search tokens that do not begin with + are OR operators, for example:

- [*A* +*B*] matches [*A*] and [*B*].
- [+*A* +*B*] matches [*A*] and [*B*].
- [*A* *B* +*C*] matches (([*A*] or [*B*]) and [*C*]).
- [*A* +*B* *C*] matches (([*A*] or [*C*]) and [*B*]).
- [*A* *B* +*C* **-***D*] matches (([*A*] or [*B*]) and [*C*] and not [*D*]).

5.9.3 Searches that include the escape character

The backslash character (\) is an escape character for the asterisk (*), question mark (?), plus sign (+), minus sign (-), single quotation mark ('), double quotation marks ("), and backslash (\), for example:

- [***] matches any string with the literal * in it.
- [+A* +*\+*] matches all strings that start with A and that contain +.

5.9.4 Advanced searching: Combining wildcards

For advanced search value fields that are numeric, such as variable searching for fields with separate physical or logical lengths, you can enter multiple values and ranges separated by blanks, for example:

- [10 20] matches 10 or 20.
- [<3 >10] matches less than 3 or greater than 10.
- [<=3 >11] matches (less than or equal to 3) or greater than 11.
- [<3 >=11] matches less than 3 or (greater than or equal to 11).
- [1-4 45] matches (between 1 and 4) or 45.

> **Important:** Combining wildcard searches is critical to the successful use of RAA in general-purpose code analysis - especially for complex large applications (also known as “production”)

Often the process of analysis begins by identifying the variation in the names that are used to refer to the component that needs to change. This situation is especially true of data elements. Because naming standards are not always followed, you must discover what variations have been used. For example, part number might be written as PARTNO, PART-ID, PART-NO, PART-NUM, PTNBR, PT-NO, and so on.

A simple text scanning tool, even using a general-purpose editor, such as ISPF, requires multiple searches, one search for each variation. Why? Because if you use partial names, such as ‘PART’, a simple text scan will also return matches to strings, such as PARTNER and PARTIAL. These unwanted matches, which are known as false positives need to be assessed and eliminated, which is typically a manual operation. With the size and scope of production systems where the number of variables can easily reach tens of millions, this elimination is an onerous prospect.

Now, we continue by implementing these search possibilities with common usage patterns for maintaining z/OS applications.

5.9.5 Searching for MVS assets conforming to a wildcard text pattern

The first or highest level application of searching is to utilize the wildcards at the level of all MVS assets that have been inventoried. This level is useful when naming conventions for a line of business conform to wildcard text patterns and you need to look across all code assets from an enterprise view.

Follow these steps:

1. Enter the wildcard text, as shown in Figure 5-42 on page 262, and click **Go**.
2. After completing the search, you get one of these results:
   – A zero Results value denotes that nothing in that asset category matched your search text.
   – If a Results value is greater than zero, by clicking the Results value, you are taken to a List page for that asset type.

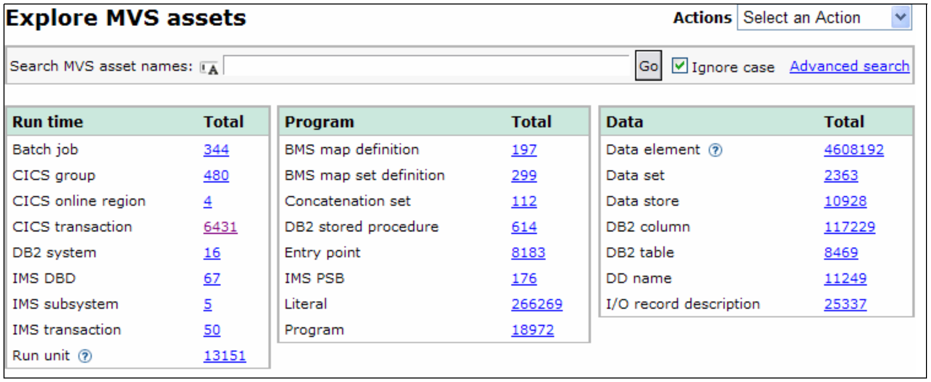


Figure 5-42 Explore MVS assets: Showing results for all searched assets that begin with PA

5.9.6 Advanced data element patterns

It is probably more common in maintenance tasks to search for data elements, such as program variables, than any other single metadata element type. Fortunately, Rational Asset Analyzer has a number of options for working with data elements (Figure 5-43).

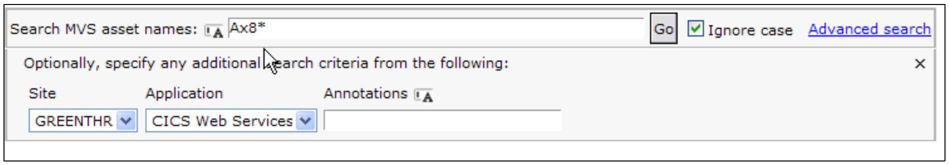


Figure 5-43 Advanced search options for a data element

As you can see in Figure 5-42, there can be hundreds of thousands or actually millions to tens of millions of variables in an enterprise inventory. So, the ability to find the exact, manageable subset is critical to your work. You can use the Advanced search dialog for Data elements, as shown in Figure 5-44 on page 264.

Look closely at Figure 5-44 to see the following filtering options:

- Search text: Your wildcard search pattern
- Site/Application: The highest level breakout, by inventoried collection
- Annotations: Custom text entries that are associated with data elements
- Program: Search within programs conforming to a wildcard search pattern
- Data type: Select one or more variable data types as additional filters:
  - CHAR: In COBOL PIC X
  - FD: Variables in the file description
  - GROUP: Filter out elementary variables
  - Numeric: In COBOL PIC 9
  - Pointer:
    - In COBOL, the LINKAGE SECTION variables
    - In PL/I and Assembler, the pointer variables
  - RD: Variables in the Report Definition section
  - SD: Variables in the Sort Description section
- Physical length: Filter the variable result set by a specific physical (byte) length
- Logical length: Filter the variable result set by a specific picture clause
- Scale: Filter the variable result set by the size of the digits to the right of the decimal point

By combining the wildcard search patterns with one or more of these filters, you can slice into a data element set numbering in the millions, with great precision. But, there are other common data element search patterns.

5.9.7 Searching in a program for data elements only referenced in PROCEDURE DIVISION

Figure 5-44 shows the program's data element summary optional filters.

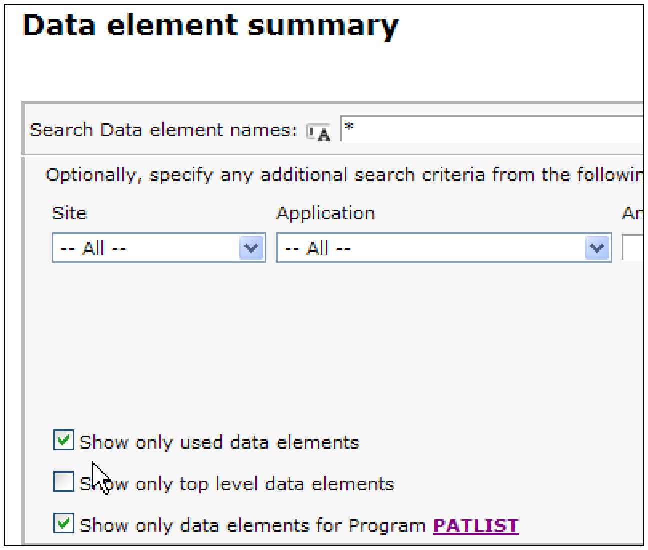


Figure 5-44 The program's data element summary optional filters

It is common in business programming to use copybooks and includes to organize and manage related variables. This method vastly simplifies many aspects of development and maintenance. But one disadvantage to this method is that each program typically only uses a small percentage (sometimes only one) of the variables in each copybook/include file.

Without Rational Asset Analyzer, this kind of research to filter out any data element in a search results list that is not referenced in a program's procedural code (any language) is painfully slow and error-prone. You must validate each entry with a secondary search into the code's methods, paragraphs, and functions.

Let us now take a look a few additional "search for" use cases or patterns.

5.9.8 Literal or constant search

Assume that you need to find the numeric literal 80 that is used throughout an application to represent a numerical factor in a computation. Imagine that if you use the ISPF 3.14 or 3.15 search facilities, you end up returning code with 80 in picture clauses, as part of paragraph/section and variable names, throughout the comments.

With Rational Asset Analyzer's search, you enter the number 80 in the Search MVS asset names under Explore MVS assets and click Go. Figure 5-45 shows what is returned, the results of the search next to all of the MVS assets in the repository.

In Figure 5-45, you can see that there is only one literal 80, as expected, by clicking the results and drilling down through the Rational Asset Analyzer summary. On the details page, you can see the metadata about the literal 80, as shown in Figure 5-46.

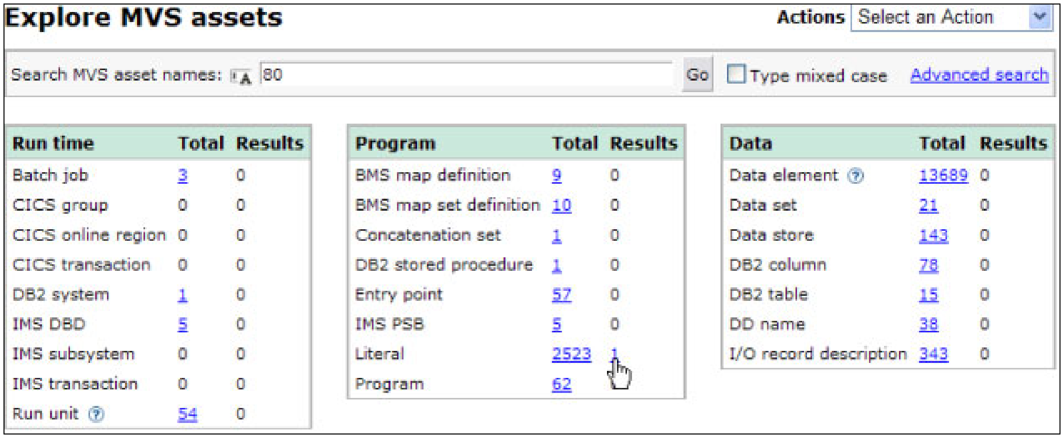
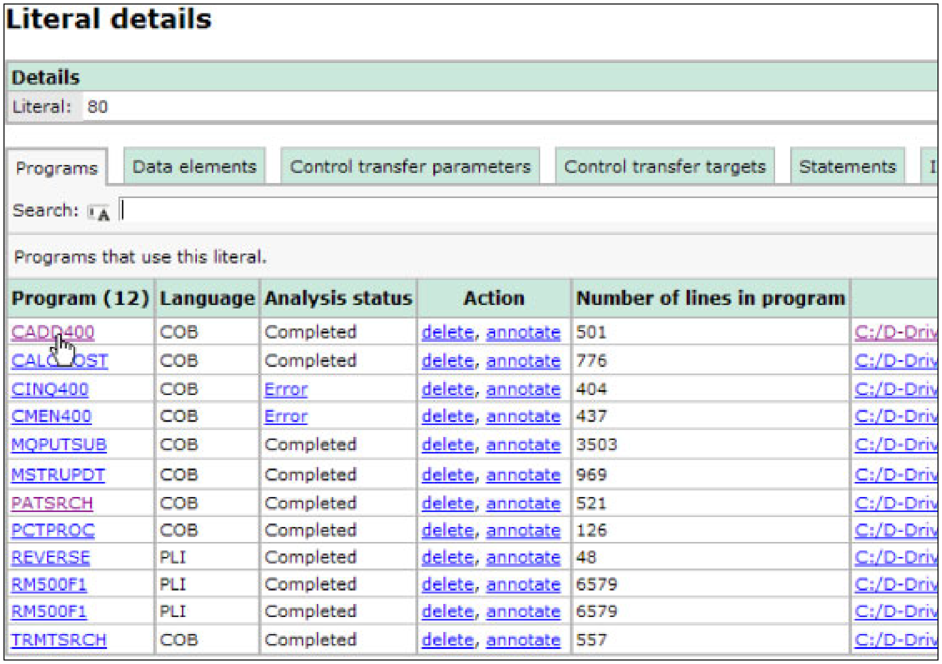


Figure 5-45 Search results for the numeric literal 80



Literal details

Details

Literal: 80

Programs | Data elements | Control transfer parameters | Control transfer targets | Statements | I

Search:

Programs that use this literal.

Program (12)	Language	Analysis status	Action	Number of lines in program	
CADD400	COB	Completed	delete, annotate	501	C:/D-Driv
CALCCOST	COB	Completed	delete, annotate	776	C:/D-Driv
CINQ400	COB	Error	delete, annotate	404	C:/D-Driv
CMEN400	COB	Error	delete, annotate	437	C:/D-Driv
MQPUTSUB	COB	Completed	delete, annotate	3503	C:/D-Driv
MSTRUPDT	COB	Completed	delete, annotate	969	C:/D-Driv
PATSRCH	COB	Completed	delete, annotate	521	C:/D-Driv
PCTPROC	COB	Completed	delete, annotate	126	C:/D-Driv
REVERSE	PLI	Completed	delete, annotate	48	C:/D-Driv
RM500F1	PLI	Completed	delete, annotate	6579	C:/D-Driv
RM500F1	PLI	Completed	delete, annotate	6579	C:/D-Driv
TRMTSRCH	COB	Completed	delete, annotate	557	C:/D-Driv

Figure 5-46 Literal details page shows all of the programs that contain the literal 80 in the repository

5.10 The "where used" search for related and dependency metadata in your applications

The search for dependent or related assets goes according to the stated metadata model captured in the repository (Figure 5-47). A series of these repository model diagrams depict Rational Asset Analyzer's Help System. Follow these steps to get to the Help System:

1. Start from the Rational Asset Analyzer Home page.
2. Click any link in the Task Help bar (Figure 5-47).
3. From the Help page, click **Administrative information** → **Database tables** → **Database tables by model**.
4. Click any of the model categories.

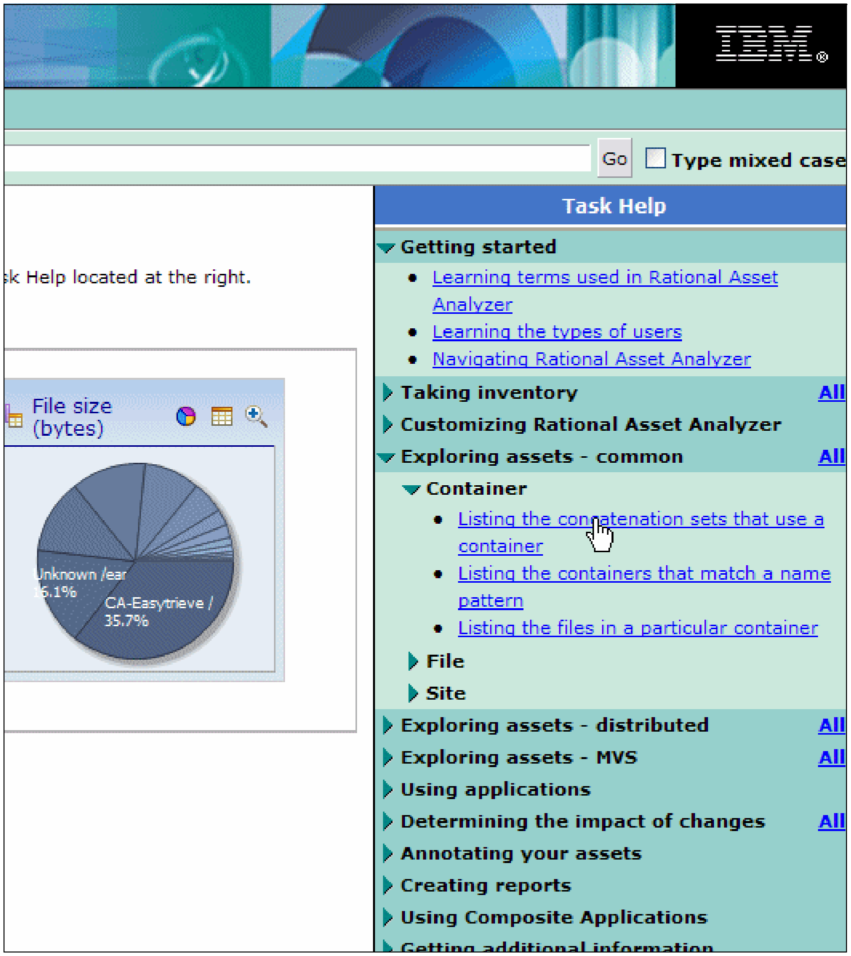


Figure 5-47 Getting to the Rational Asset Analyzer Help System from the Rational Asset Analyzer home page

5.10.1 Dependency diagrams

You can think of the metadata models that are shown in these diagrams as "*dependency diagrams*". They depict the systemic dependencies of z/OS software as captured through the Rational Asset Analyzer inventory collection.

Figure 5-48 depicts the DB2 objects. You can see that Rational Asset Analyzer stores information in its repository as each DB2 subsystem is captured. Each subsystem can have one to many tables. Each table has its own metadata and can have one to many columns and view dependencies.

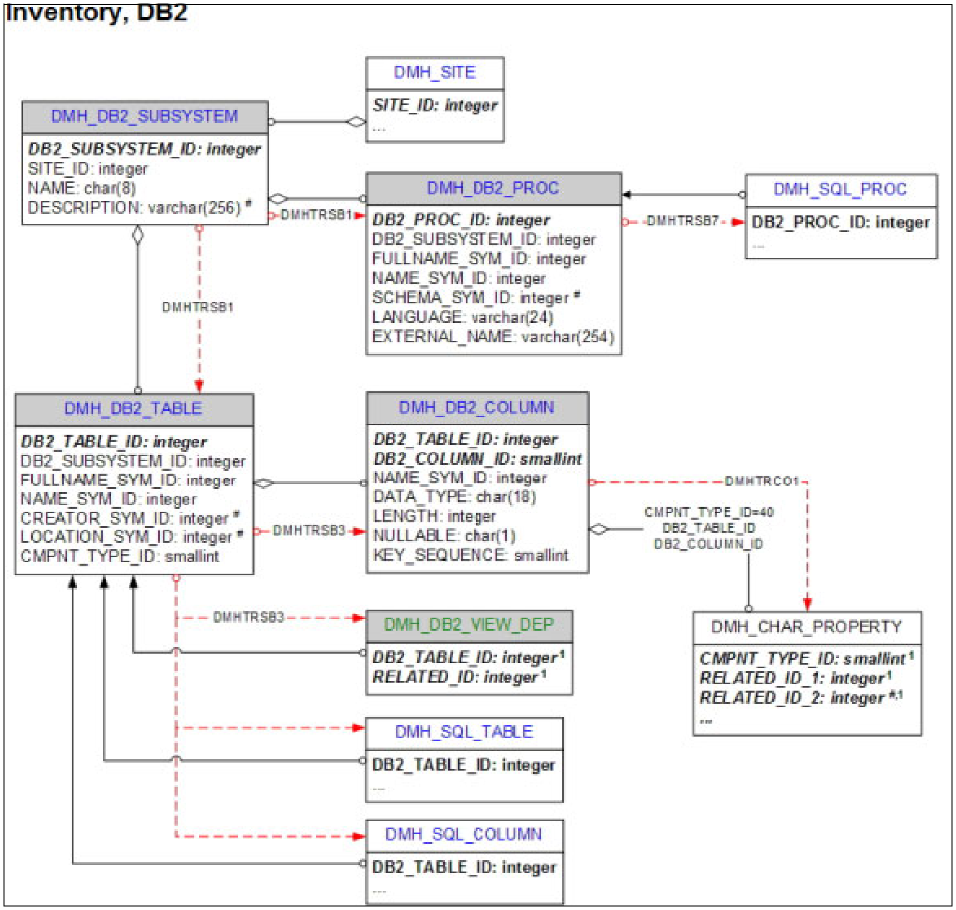


Figure 5-48 The Rational Asset Analyzer metadata that is maintained about DB2 objects

As another example, Figure 5-49 shows the dependency model for z/OS batch jobs, showing the jobs, job steps, and DD statements. With the depth and through the rich metadata contained in the repository, you can use Rational Asset Analyzer to answer the vast majority of the “where used” questions.

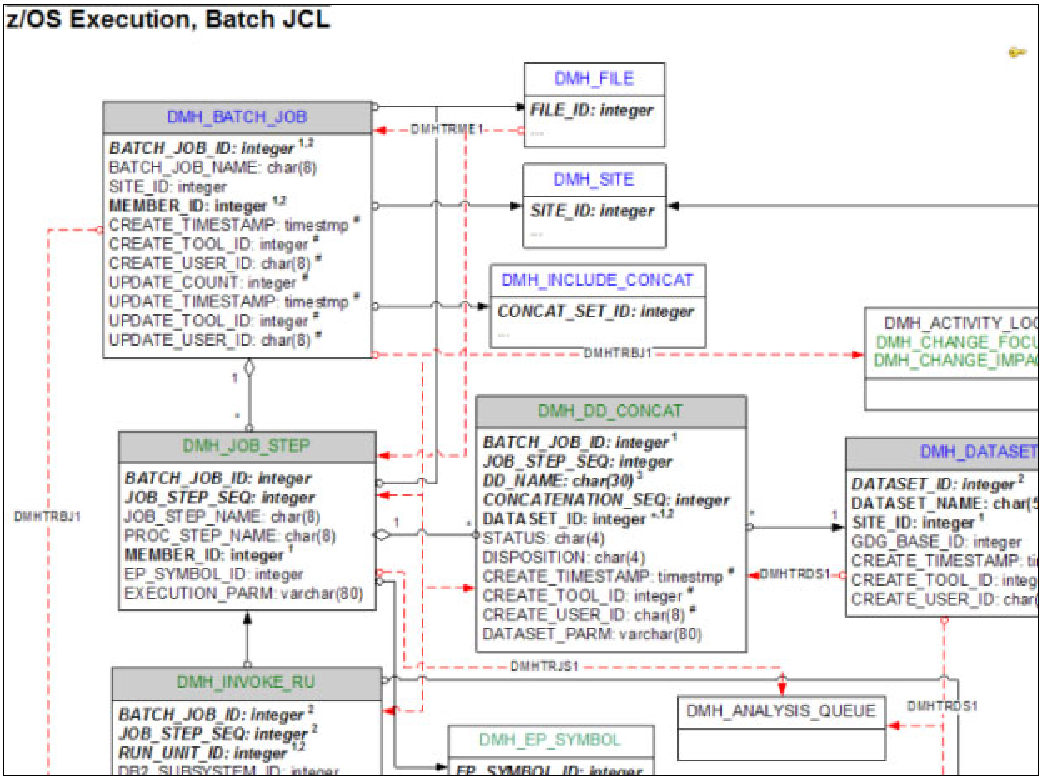


Figure 5-49 The Rational Asset Analyzer z/OS Execution, Batch JCL metadata repository model

Next, we look at a few specific examples or usage patterns of dependency searches:

- What DB2 tables are referenced in a batch job?
- What is the extended called/calling chain from a batch job?
- What files, databases, and windows are used in a CICS transaction?

Each example starts from having already completed a search to get to either a Rational Asset Analyzer summary or detailed page. We will not duplicate the exact search steps here, because they are relatively simple and most often a search from the Explore MVS assets page.

5.10.2 DB2 tables referenced in a batch job

In order to assess the impact of proposed changes to DB2 table schema, it is important to be able to determine which batch jobs might be affected. Of course, there are many other effects to assess, which we discuss in the next section, but to scope out the affected batch jobs can be problematic:

- There is no direct correlation of batch jobs to DB2 tables in the DB2 catalog.
- DB2 tables are not referenced with DD statements in the way that VSAM/QSAM files are in batch JCL.

What to do? Well, we have already seen one way that Rational Asset Analyzer represents the relationship between DB2 tables and batch jobs in the Batch job diagram (Figure 5-31 on page 249 and Figure 5-31 on page 249).

Diagrams are extremely useful for "what is" learning and discovery analysis; however for "where used" analysis, we might prefer a list that can be saved to project specifications or work tasks. You can create this type of list by using a custom query, which can be invoked, as shown in Figure 5-50 on page 271, and which returns a list of tables, as shown in Figure 5-51 on page 272.

5.10.3 What is a custom query

We explain custom queries in-depth later in 5.12, “Custom queries” on page 291. But for now, a custom query is a user-written query that accesses the information in the Rational Asset Analyzer metadata repository, which is a DB2 database, through ordinary SQL statements.

You can use custom queries to extend the predefined functionality of Rational Asset Analyzer. Rational Asset Analyzer users frequently use custom queries. It is simple to write a custom query if you have SQL coding expertise. You can export and import custom queries and associated with a Rational Asset Analyzer action, as shown in Figure 5-50 and Figure 5-51.

We have exported all of the custom queries that are used in this section, and we have placed them on the book source server for import and reuse by you. You can access them by following the instructions in Appendix B, “Additional material” on page 599.

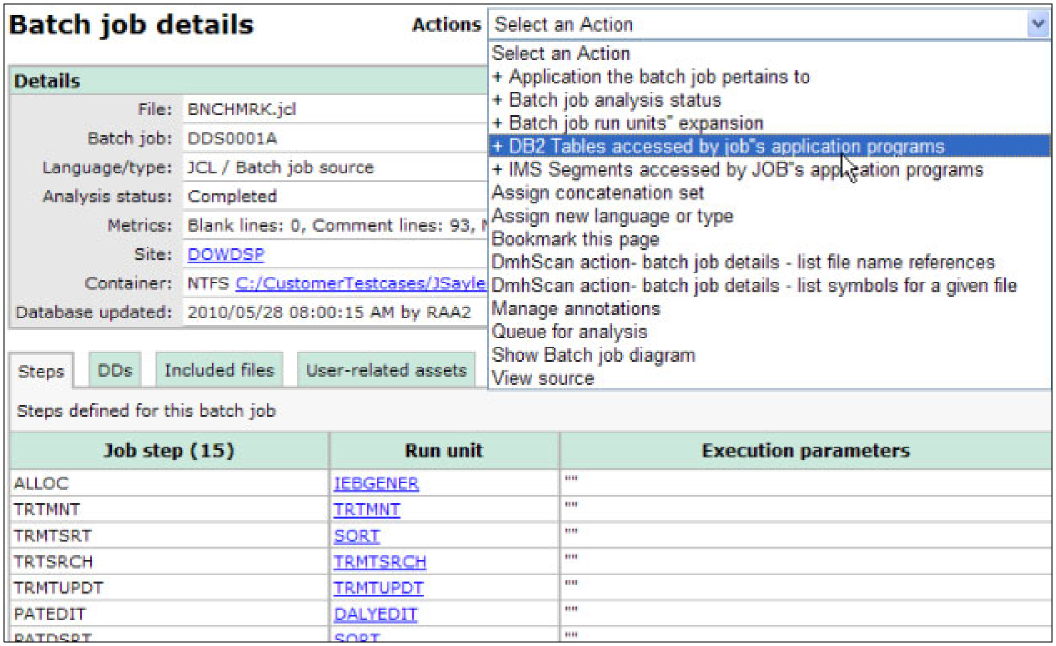
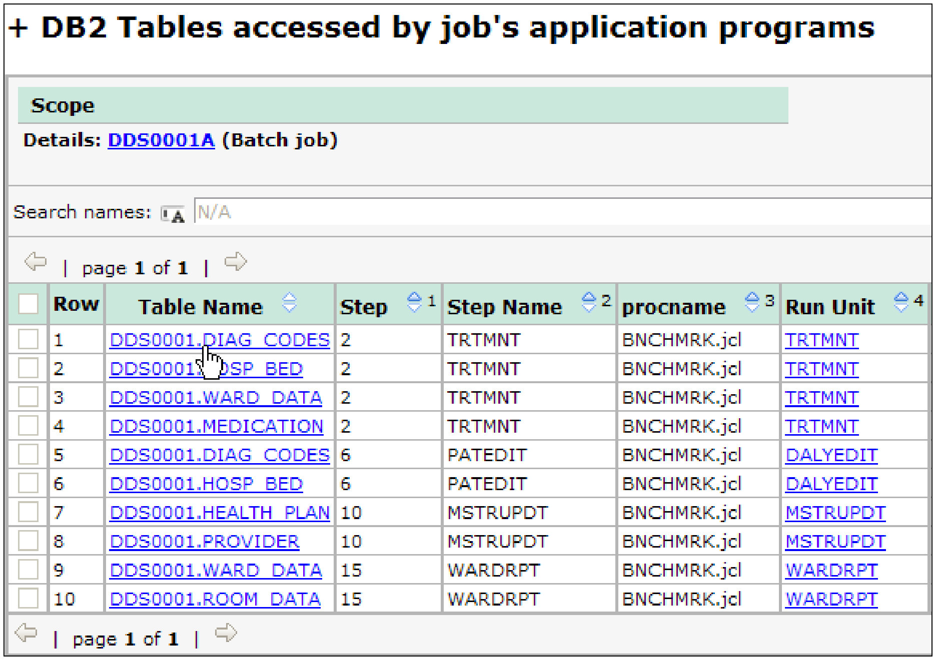


Figure 5-50 Batch job details: Invoking a custom query


+ DB2 Tables accessed by job's application programs

Scope

Details: DDS0001A (Batch job)

Search names: N/A

| page 1 of 1 |

	Row	Table Name	Step	Step Name	procname	Run Unit
	1	DDS0001.DIAG_CODES	2	TRTMNT	BNCHMRK.jcl	TRTMNT
	2	DDS0001.HOSP_BED	2	TRTMNT	BNCHMRK.jcl	TRTMNT
	3	DDS0001.WARD_DATA	2	TRTMNT	BNCHMRK.jcl	TRTMNT
	4	DDS0001.MEDICATION	2	TRTMNT	BNCHMRK.jcl	TRTMNT
	5	DDS0001.DIAG_CODES	6	PATEDIT	BNCHMRK.jcl	DALYEDIT
	6	DDS0001.HOSP_BED	6	PATEDIT	BNCHMRK.jcl	DALYEDIT
	7	DDS0001.HEALTH_PLAN	10	MSTRUPDT	BNCHMRK.jcl	MSTRUPDT
	8	DDS0001.PROVIDER	10	MSTRUPDT	BNCHMRK.jcl	MSTRUPDT
	9	DDS0001.WARD_DATA	15	WARDRPT	BNCHMRK.jcl	WARDRPT
	10	DDS0001.ROOM_DATA	15	WARDRPT	BNCHMRK.jcl	WARDRPT

| page 1 of 1 |

Figure 5-51 A custom query showing a list of DB2 tables accessed within a batch JCL job

5.10.4 Extended called/calling chain from a batch job

When you need to change a batch job or even if you are simply supporting the job in a data center or on-call position, it can be extremely useful to understand the entire called/calling chain from each step in the batch job. This understanding can help you to learn “where used” information that you might need for analysis for an ABEND resolution.

We use another custom query for this example:

- Figure 5-52 on page 273 shows the details page of any batch job and our custom query as an action.
- Figure 5-53 on page 274 shows the list of batch job steps and, for each step, the metadata that is available about called/calling programs.

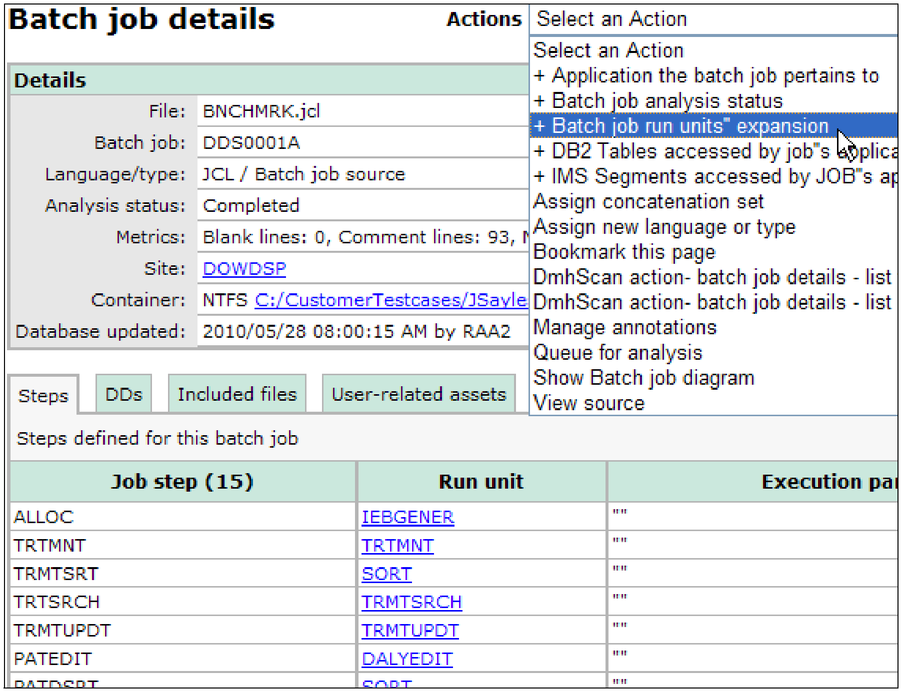
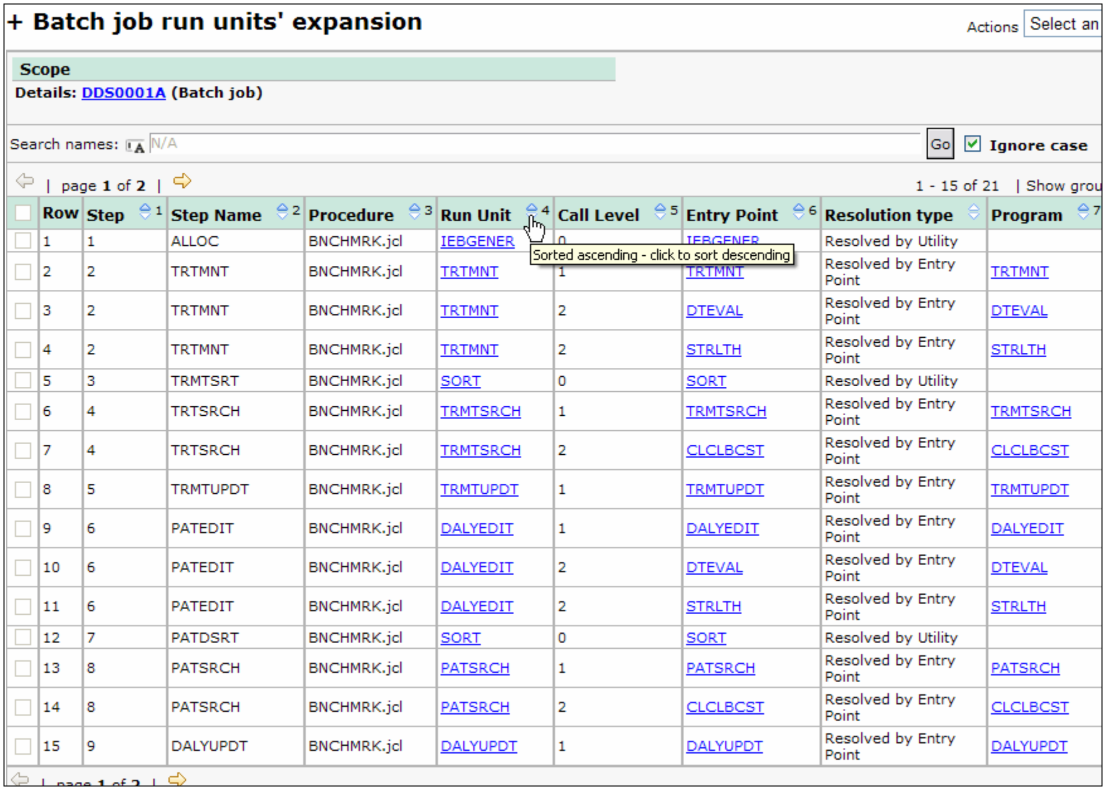


Figure 5-52 Executing a custom query to provide a list of called/calling programs for a batch job

As you can see in Figure 5-52, each job step is broken out by run unit (essentially the EXEC PGM= *load module name*). And then for each load module, all called sub-modules are documented. To explain, in the TRTMNT job step, the EXEC PGM= is the TRTMNT load module, which is a COBOL program that dynamically calls DTEVAL and STRLTH (Figure 5-53).



+ Batch job run units' expansion

Actions | Select an

Scope

Details: DDS0001A (Batch job)

Search names: N/A | Go | Ignore case

| page 1 of 2 | 1 - 15 of 21 | Show grou

Row	Step	Step Name	Procedure	Run Unit	Call Level	Entry Point	Resolution type	Program
1	1	ALLOC	BNCHMRK.jcl	IEBGENER	0	IEBGENER	Resolved by Utility	
2	2	TRTMNT	BNCHMRK.jcl	TRTMNT	1	TRTMNT	Resolved by Entry Point	TRTMNT
3	2	TRTMNT	BNCHMRK.jcl	TRTMNT	2	DTEVAL	Resolved by Entry Point	DTEVAL
4	2	TRTMNT	BNCHMRK.jcl	TRTMNT	2	STRLTH	Resolved by Entry Point	STRLTH
5	3	TRMTSRT	BNCHMRK.jcl	SORT	0	SORT	Resolved by Utility	
6	4	TRTSRCH	BNCHMRK.jcl	TRMTSRCH	1	TRMTSRCH	Resolved by Entry Point	TRMTSRCH
7	4	TRTSRCH	BNCHMRK.jcl	TRMTSRCH	2	CLCLBCST	Resolved by Entry Point	CLCLBCST
8	5	TRMTUPDT	BNCHMRK.jcl	TRMTUPDT	1	TRMTUPDT	Resolved by Entry Point	TRMTUPDT
9	6	PATEDIT	BNCHMRK.jcl	DALYEDIT	1	DALYEDIT	Resolved by Entry Point	DALYEDIT
10	6	PATEDIT	BNCHMRK.jcl	DALYEDIT	2	DTEVAL	Resolved by Entry Point	DTEVAL
11	6	PATEDIT	BNCHMRK.jcl	DALYEDIT	2	STRLTH	Resolved by Entry Point	STRLTH
12	7	PATDSRT	BNCHMRK.jcl	SORT	0	SORT	Resolved by Utility	
13	8	PATSRCH	BNCHMRK.jcl	PATSRCH	1	PATSRCH	Resolved by Entry Point	PATSRCH
14	8	PATSRCH	BNCHMRK.jcl	PATSRCH	2	CLCLBCST	Resolved by Entry Point	CLCLBCST
15	9	DALYUPDT	BNCHMRK.jcl	DALYUPDT	1	DALYUPDT	Resolved by Entry Point	DALYUPDT

Sorted ascending - click to sort descending

Figure 5-53 Custom query to provide a called/calling program chain list for each run unit in a batch job

5.10.5 Files, databases, and windows used in a CICS transaction

Just as it can be useful to determine what assets are used in a job step, it is equally important to be able to see at a glance all of the resources that are consumed by CICS or IMS transactions.

In Figure 5-54, you can see the results of a Rational Asset Analyzer details page for transaction APX1 that shows the called/calling program chain.

In Figure 5-55, you can see that the transaction contains the following dependent asset references:

- BMS map (APX1MAP)
- Five Temporary Storage (TS) queues
- Two IMS database segments
- One DB2 table

Note that this diagram is not a custom query. It is a standard Rational Asset Analyzer operation. Rational Asset Analyzer has an enormous number of common and extremely useful predefined dependency and “where used” reports in the summary and detailed pages and in their tab entries.

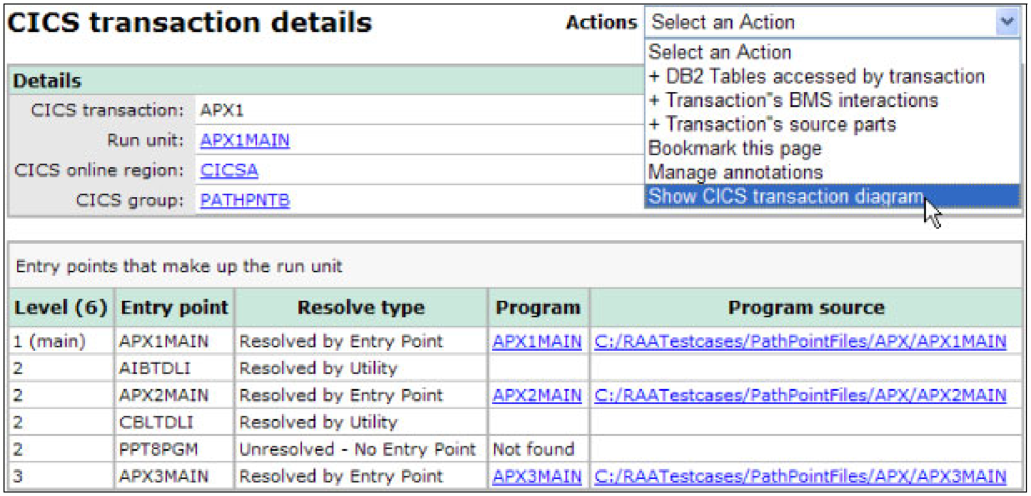


Figure 5-54 CICS transaction details window showing called/calling programs

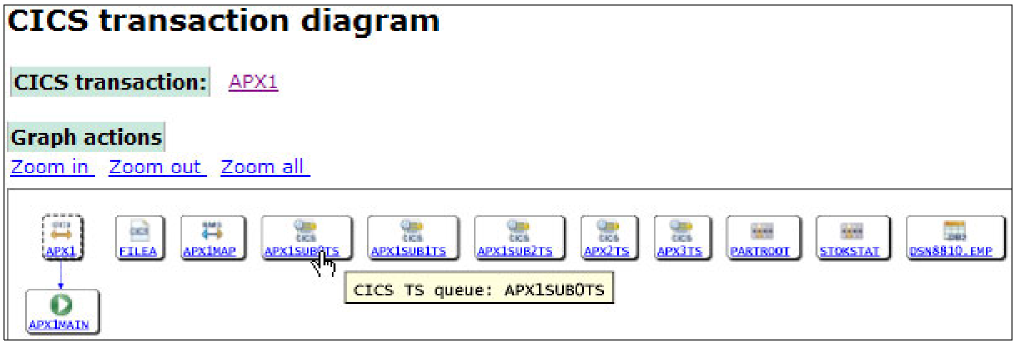


Figure 5-55 CICS transaction diagram showing all resources that are part of a CICS transaction

The same two analysis capabilities exist for IMS, as shown in Figure 5-56 and Figure 5-57.

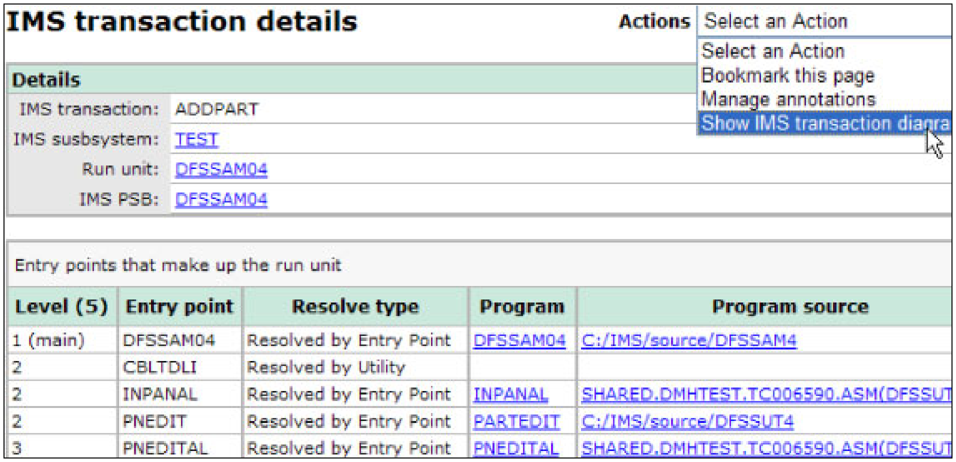


Figure 5-56 IMS transaction details page

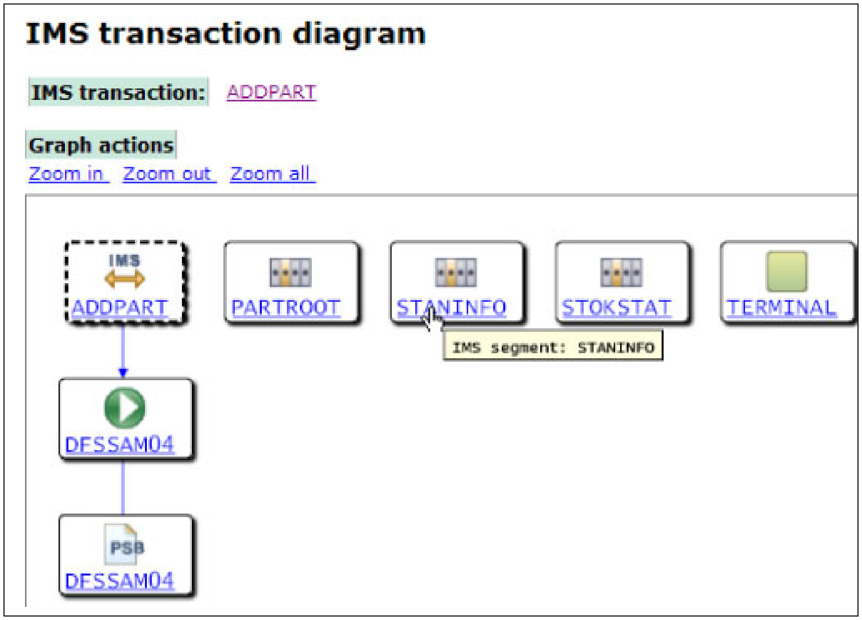


Figure 5-57 IMS transaction diagram

5.10.6 DB2 tables: "Where used and how used" analysis

After you understand where a particular asset is used and what depends on it, you naturally move down a level to analyze "how" an asset is used, especially if that asset is a file, database, or DB2 table. Rational Asset Analyzer captures, as part of its inventory process, details about the accesses to data sources and displays this information in convenient details pages.

To understand how a particular DB2 table is accessed, start by searching on the fully qualified table name, for example, `DSN8810.EMP`, or you can substitute wildcard characters. However, you must adhere to the two-part (fully qualified) naming convention: *SchemaName.TableName*.

From there (Figure 5-58), you can research the following information through hyperlinks and metadata results:

- Each program that accesses the table and see the access type
- The table's column names and views
- Where the table declaration occurs in the program source, which matters if you have to update a table column and change the corresponding COBOL Picture clause

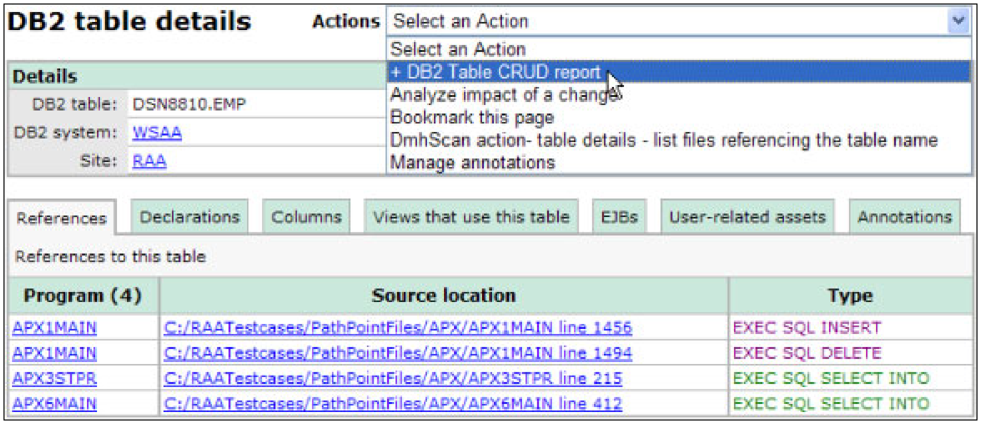


Figure 5-58 DB2 table "where used/how used" details

Figure 5-58 on page 277 also shows that a custom query exists to produce a Create/Read/Update/Delete (CRUD) matrix report about the table, as shown in Figure 5-59. This type of information can be vital to help you perform the following types of analyses:

- Locking and contention issues
- Normalization and de-normalization redesigns
- Index creation
- Runtime performance issues

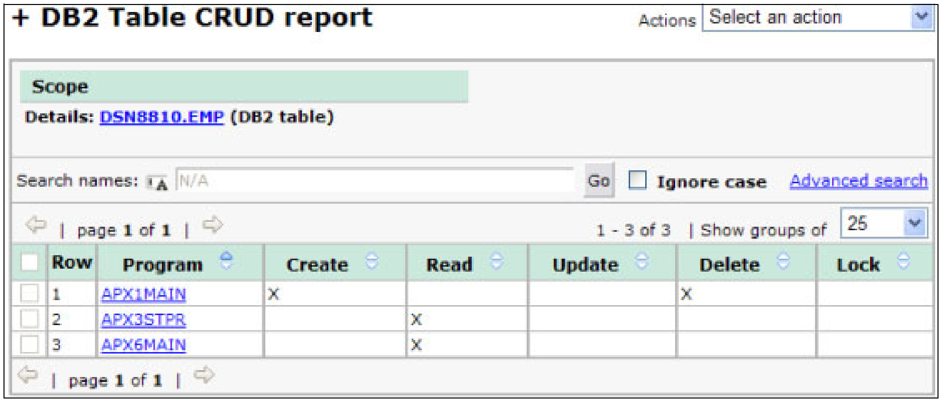


Figure 5-59 DB2 Table CRUD report analysis matrix: A custom query

5.11 Impact analysis

An *impact analysis* refers to the kind of deep analytic research that is necessary before making a change of any consequence to an existing software application. Before you modify a single line of working production code, you need to analyze the effect of change across all potentially affected software in your application and in any other application that uses any of the affected software components. An example was the Y2K problem. Impact analysis forms the basis of many analysis day-to-day tasks. We perform impact analysis for several reasons:

- Changes to production applications occur on a regular basis.
- The larger or more complex the application or the more pervasive the proposed change, the more involved, time-consuming and error-prone the impact analysis becomes.
- This complexity increases the time to market, expense, and risk, which, given the potential number of elements to analyze, is not surprising.

Impact analysis without Rational Asset Analyzer

Performing impact analysis manually involves performing repetitious text scans in this order:

1. Searching for affected assets
2. Saving interim results in text format for analysis
3. Examining each potentially affected asset for additional references and clues to what you must change
4. Starting back at step 1, searching through the same source files for each potentially affected asset
5. Creating the list of all assets that need to be changed

Rational Asset Analyzer's impact analysis feature provides a significantly faster way to gather this information. Using the parsed inventory in the repository, Rational Asset Analyzer searches all known dependencies by using proven software engineering formulas that do not depend on field naming conventions. And using the metadata captured, Rational Asset Analyzer creates a number of useful reports that can greatly shorten your time to task completion.

Performing Impact Analysis

Rational Asset Analyzer supports a wide variety of Impact Analysis "seeds" or starting points, including but not limited to performing Impact Analysis on these objects:

- One or more variables in a copybook
- One or more variables coded in-line within a program
- A range of statements in a program in the PROCEDURE DIVISION
- A VSAM or QSAM file
- A DB2 table or table column

5.11.1 Impact Analysis: Increasing a key field's declaration size

Consider this example. The application for which you are responsible has outgrown the built-in limitations of an identifier's declaration (a patient identification field). To accommodate this change, you have been asked to estimate the impact of increasing the length of the patient identification number from six positions to eight positions.

To estimate the impact of this change, you need to perform these tasks:

1. Find the data names that are used for the patient identification number.
2. Find which application components will be affected by the change.
3. Estimate how long it will take to make the changes.
4. Plan developer work assignments.

As we step through this process, assume that you have already inventoried and successfully parsed the application. This simple example shows the use of Impact Analysis starting with a single field and a single search phrase, which is often the case especially in newer systems. It is equally common for your searches to begin with a complex search pattern set, which is described in 5.9.4, “Advanced searching: Combining wildcards” on page 261, to discover as many references to the data on which you are basing your analysis as possible when variable declarations have been coded without the benefit of strong and consistent naming convention management.

Figure 5-60 shows the entire process for Impact Analysis:

- Starting with identifying the elements that you want to include in your Impact Analysis, you perform the Impact Analysis operation. The Impact Analysis operation is a wizard-driven process. At the end of the process, your Impact Analysis results are written to the Rational Asset Analyzer metadata repository (DB2 tables) and stored for reuse.
- You then review the Impact Analysis report output, making use of the found dependencies in various ways, including the detailed task assignments.
- Finally, you can at any time return to the stored Impact Analysis and revisit any aspects of it, because all of the results of the analysis are available in the repository.

It is important to note that when reexamining a stored Impact Analysis, you are viewing the *stored* results, not rerunning the entire Impact Analysis. This approach is useful for at least two reasons:

- The Impact Analysis represents a point-in-time snapshot of the state of your application, which you can compare against other stored impact analyses over time to assess the historical trends and health of your application
- Running Impact Analysis in large applications can be time-consuming. Reviewing stored Impact Analysis reports is not time-consuming.

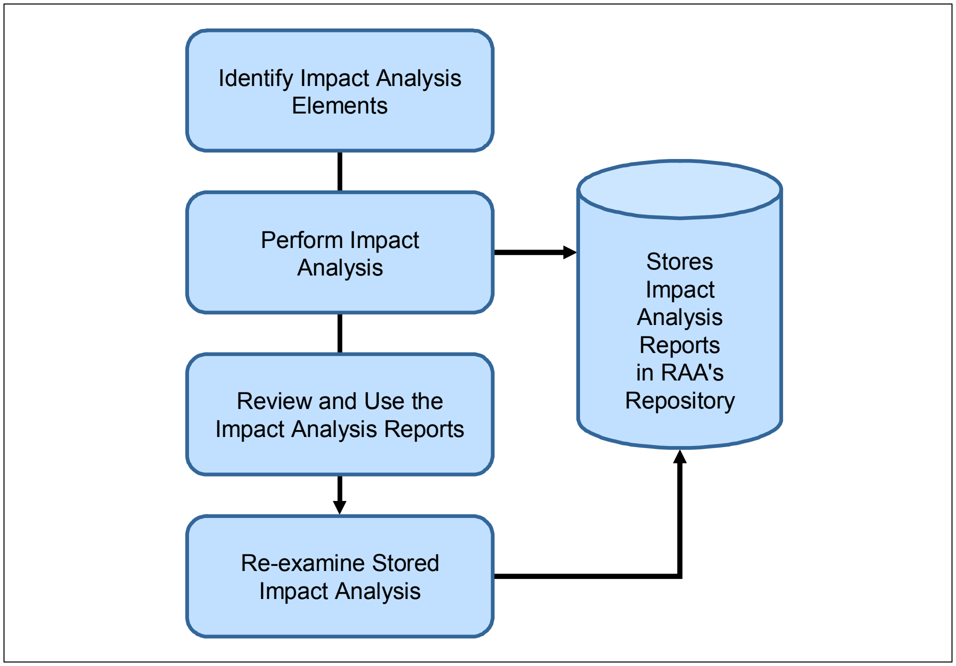
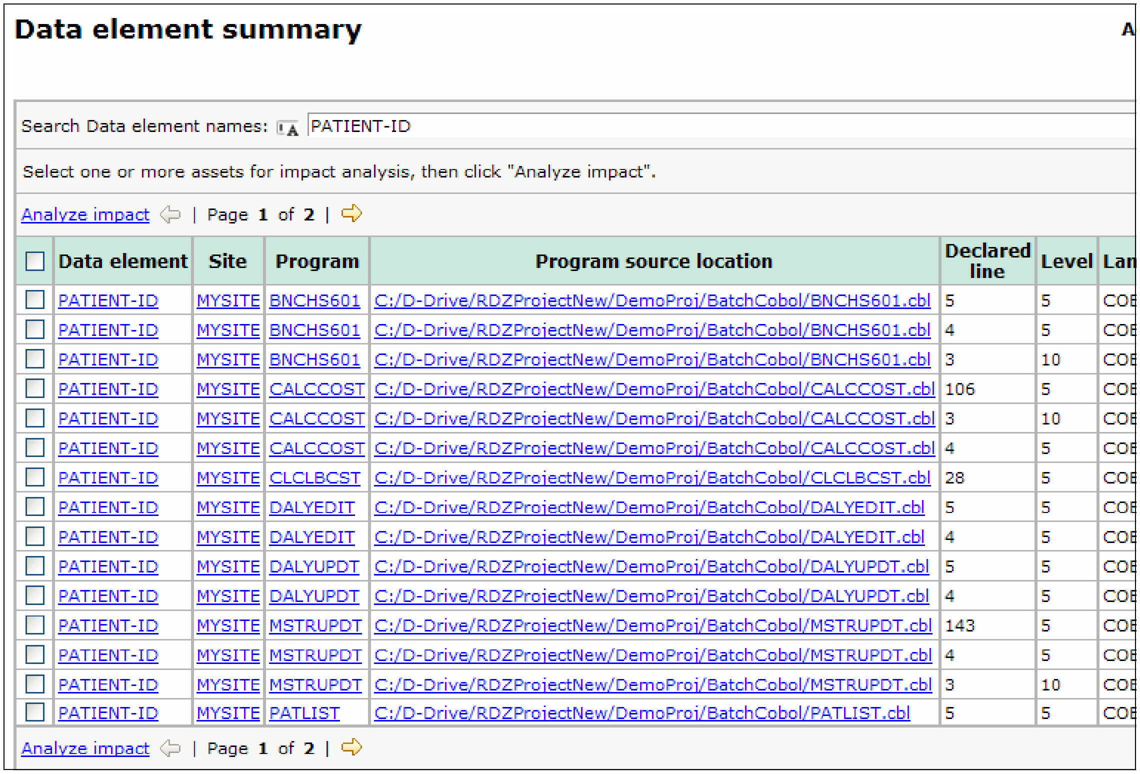


Figure 5-60 The Rational Asset Analyzer Impact Analysis process

Starting an Impact Analysis process

We continue by describing a sample Impact Analysis process, starting with Figure 5-61 on page 282, selecting the element to be analyzed. From this, you can see that it is easy to begin the Impact Analysis process:

1. From a list (summary) page for any element that can be impact analyzed, you select the element's check box and click **Analyze impact**.
2. From any details page for an element that supports Impact Analysis, you select **Analyze the impact of a change** in the Actions drop-down menu. This step opens the Create an Impact Analysis wizard, as shown in Figure 5-62 on page 283.



Data element summary

Search Data element names: PATIENT-ID

Select one or more assets for impact analysis, then click "Analyze impact".

Analyze impact | Page 1 of 2 |

	Data element	Site	Program	Program source location	Declared line	Level	Lan
☐	PATIENT-ID	MYSITE	BNCHS601	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/BNCHS601.cbl	5	5	COE
☐	PATIENT-ID	MYSITE	BNCHS601	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/BNCHS601.cbl	4	5	COE
☐	PATIENT-ID	MYSITE	BNCHS601	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/BNCHS601.cbl	3	10	COE
☐	PATIENT-ID	MYSITE	CALCCOST	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/CALCCOST.cbl	106	5	COE
☐	PATIENT-ID	MYSITE	CALCCOST	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/CALCCOST.cbl	3	10	COE
☐	PATIENT-ID	MYSITE	CALCCOST	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/CALCCOST.cbl	4	5	COE
☐	PATIENT-ID	MYSITE	CLCLBCST	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/CLCLBCST.cbl	28	5	COE
☐	PATIENT-ID	MYSITE	DALYEDIT	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/DALYEDIT.cbl	5	5	COE
☐	PATIENT-ID	MYSITE	DALYEDIT	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/DALYEDIT.cbl	4	5	COE
☐	PATIENT-ID	MYSITE	DALYUPDT	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/DALYUPDT.cbl	5	5	COE
☐	PATIENT-ID	MYSITE	DALYUPDT	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/DALYUPDT.cbl	4	5	COE
☐	PATIENT-ID	MYSITE	MSTRUPDT	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/MSTRUPDT.cbl	143	5	COE
☐	PATIENT-ID	MYSITE	MSTRUPDT	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/MSTRUPDT.cbl	4	5	COE
☐	PATIENT-ID	MYSITE	MSTRUPDT	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/MSTRUPDT.cbl	3	10	COE
☐	PATIENT-ID	MYSITE	PATLIST	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/PATLIST.cbl	5	5	COE

Analyze impact | Page 1 of 2 |

Figure 5-61 Launching an Impact Analysis operation

3. All Impact Analysis operations are saved in the Rational Asset Analyzer repository. As part of the save process, you can name the Impact Analysis (Figure 5-62 on page 283). The default name is a Rational Asset Analyzer-generated qualified string (in our case: `TRTMNT:PATIENT-ID`). Note that you do not necessarily have to remember this name, because when you want to reexamine a saved Impact Analysis, all of your saved analyses are shown in a list. However, it is probably a good practice to uniquely name, as well as carefully describe, your Impact Analysis to make it simpler to reuse.

   Also, note about this wizard:

   – You can cancel the Impact Analysis at any time by simply closing the wizard or clicking **Cancel**.

   – By clicking **View**, you see each of your Impact Analysis seeds (a variable, statement range, file, or so forth) from a Rational Asset Analyzer Details page.

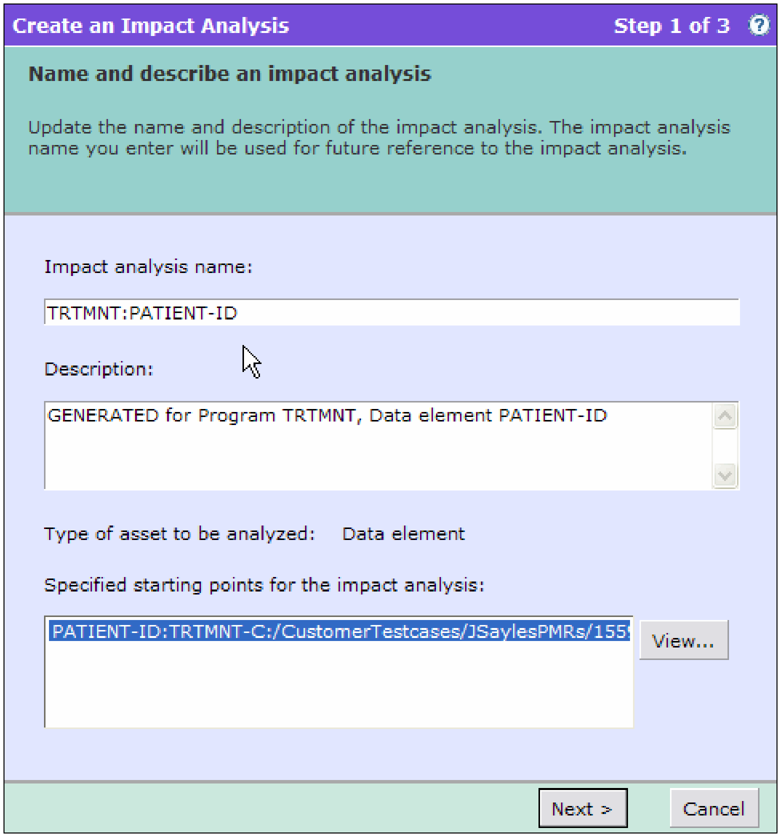


Figure 5-62 The first step in the Impact Analysis operation: Name and describe your Impact Analysis

4. In Figure 5-62, you continue the Impact Analysis process by clicking **Next**. We do not show all of the options in this book, but note that on the additional wizard steps, you can further customize the Impact Analysis with these options:
   – Starting points: If what you are impact analyzing is part of a batch job or transaction, you can specify at what job or transaction to begin the operation
   – Scope: Allows you to extend the Impact Analysis to a particular inventoried application
   – Number of levels to impact analyze to: Allows you to specify a limit to the number of indirect references to your initial seed element as a defined level to the Impact Analysis

These options allow you to customize the Impact Analysis operation or, when dealing with extremely large applications, to limit the Impact Analysis in regard to time and results details.

After you have completed running the Impact Analysis wizard

The Impact Analysis process begins in the background. When it is complete, you click **Finish**, which displays the Impact analysis details page shown in Figure 5-63.

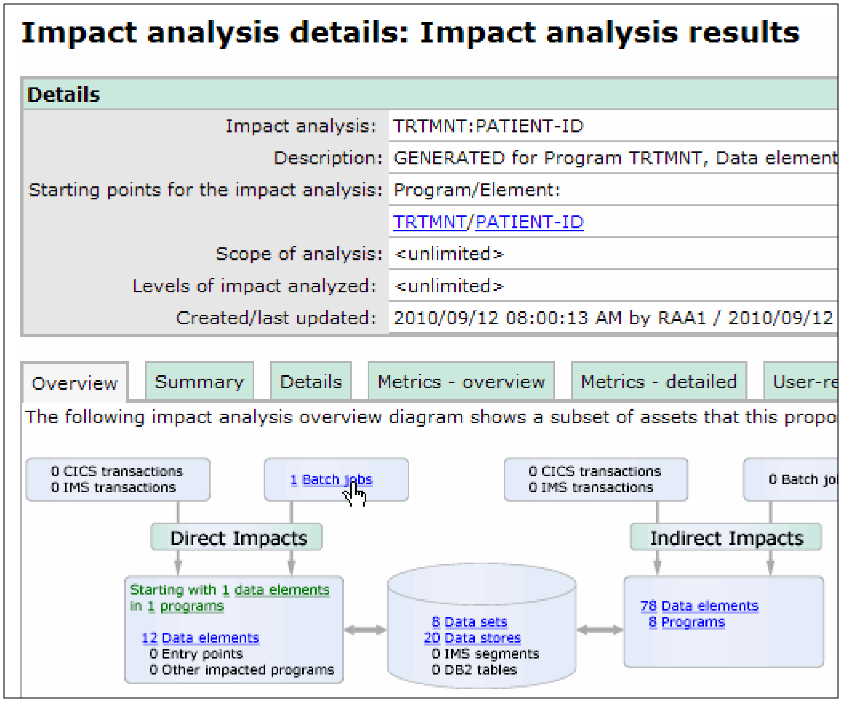


Figure 5-63 A completed Impact Analysis showing the Overview tab

In Figure 5-63, you can see a graphical depiction of the Impact Analysis results, breaking out the findings into categories, metadata element type (CICS transactions, IMS transactions, programs, data elements, data sets, and so forth). For each element with a non-zero number as a prefix (in Figure 5-63, see 12 Data elements), Rational Asset Analyzer creates a hyperlink to a summary (list) page with links to the individual elements that are part of the Impact Analysis results set. You can also see from Figure 5-63 that the entire set of elements is broken into Direct Impacts and Indirect Impacts.

5.11.2 Impact Analysis: Operation - direct and indirect impacts

Direct impacts are those components that reference the starting points (seed search values) of the analysis. *Indirect impacts* are those components that might be impacted, because they have a data source in common. The diagram shown in Figure 5-63 on page 284 provides two of the inputs for estimating project size and cost: the number of affected programs and the number of affected data elements.

During this Impact Analysis, direct impacts were determined by identifying the programs that referenced the DB2 columns within their logic, and by tracing the data flow to and from these columns to other data elements within the program logic (or to data elements passed to called programs). Rational Asset Analyzer stops the data flow tracing when one of the following conditions occurs:

- A literal data definition is found.
- The first program in a *run unit* (a group of programs that call one another) is reached.
- A file or database is accessed in the program logic.

Programs are then traced to the transaction and jobs that use them. Indirect impacts were determined by identifying the I/O record definitions that map to the same byte location in those files and databases identified in the direct impacts, identifying the programs (not already identified) that use those I/O record definitions, and identifying the transactions and jobs that use these programs.

Indirect impacts suggest components that might also be affected by the proposed change, because they have data in common. They must be investigated further. For example, I/O record definitions that are used by programs listed as indirect impacts might reveal new data element names that can be the starting point for additional impact analyses.

5.11.3 Impact Analysis: Summary tab

Figure 5-64 shows the Summary tab of the Impact analysis details: Impact analysis results page.

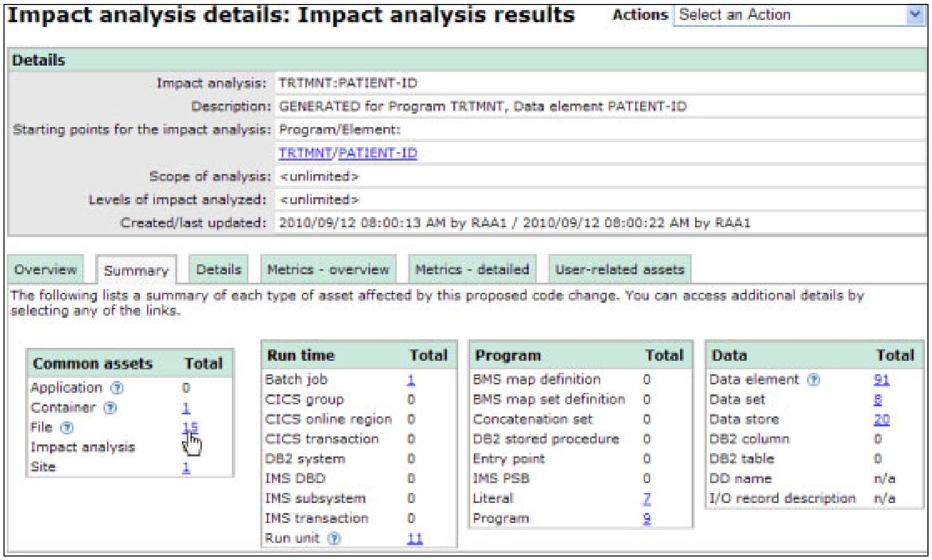


Figure 5-64 Impact analysis: Summary tab

The Summary tab (Figure 5-64) shows the information discovered during the Impact Analysis operation across all of the categories of Rational Asset Analyzer metadata supported in the repository. As in the Overview tab, you can click the hyperlinks in the Summary tab to list the individual elements to be analyzed.

5.11.4 Impact Analysis: Details tab

The Details tab (Figure 5-65) provides a low-level, element-level report of every directly and indirectly impacted element within the scope of your report. This level of detail is extremely precise, and this report is often copied exactly to either a project management reporting tool or another organizational package, such as Microsoft Excel, to be used as work item tasks, assignments, and developer/analyst follow-up activities.

By copying and pasting the information in the Details tab, you can also copy and paste the URLs (hyperlinks), so that developers who are using a spreadsheet can click the link in the spreadsheet and open Rational Asset Analyzer to the specific Details page that was generated by the Impact Analysis report.

You can do this in one of two ways:

1. Right-click over your browser while showing the Details tab, and select **Export to Microsoft Excel**.

2. Select any subset of the Details report by using one of these methods (this report can be extremely long, depending on the scope of the Impact Analysis results set):

   – Drag your mouse through the relevant portions of the report and copy the text (web page headers and detailed links).

   – Open an Excel spreadsheet and paste the subset into rows.

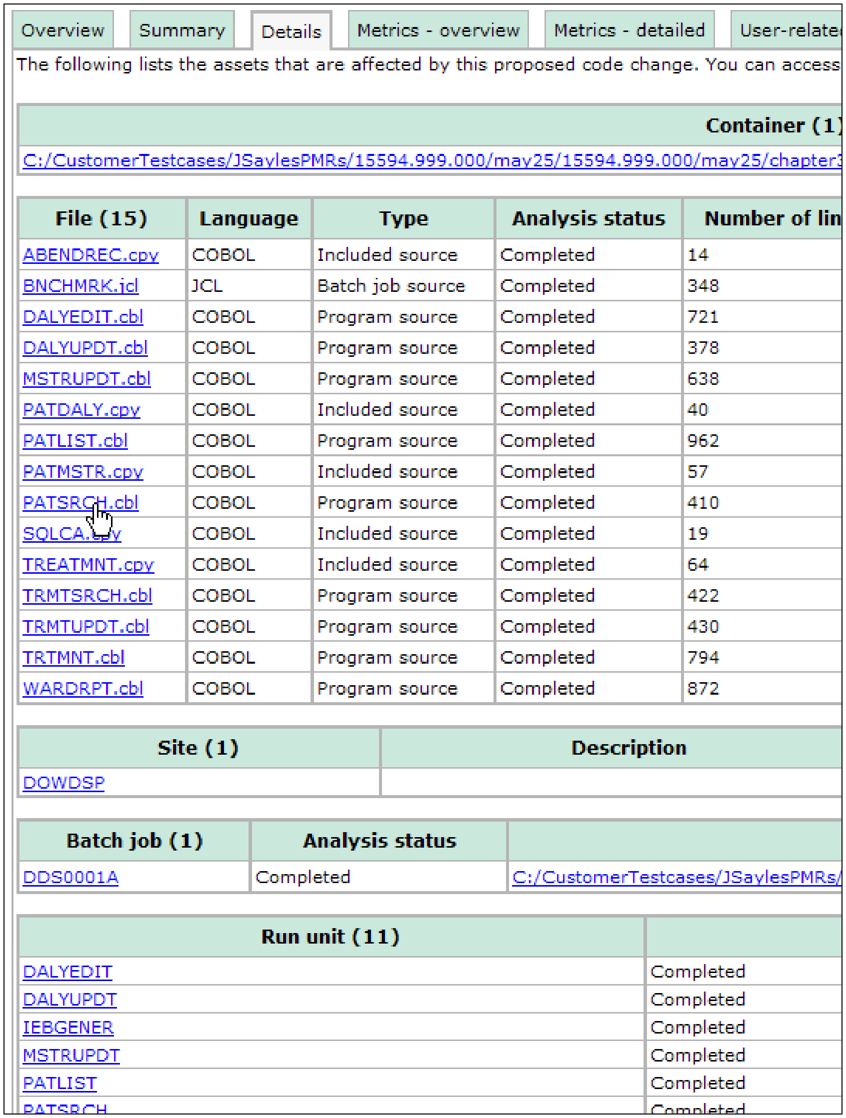


Figure 5-65 Impact analysis: Details tab

5.11.5 Complexity metrics generated by an Impact Analysis

Figure 5-66 and Figure 5-67 show several of the metrics reports that are generated by Impact Analysis. This information is the same type of complexity/cost-analysis data points that were covered in 5.6, “Rational Asset Analyzer and source metrics” on page 234.

The difference here, however, is that the context of these values is specific to the file changes for *only* the source code that is impacted by a change, instead of the metrics for the entire metadata repository. In this way, the information or complexity costs shown can help you to budget for your application modification.

The Metrics - detailed tab presents the same metrics but broken down further into language elements, such as COBOL, Assembler, PL/I, and so forth.

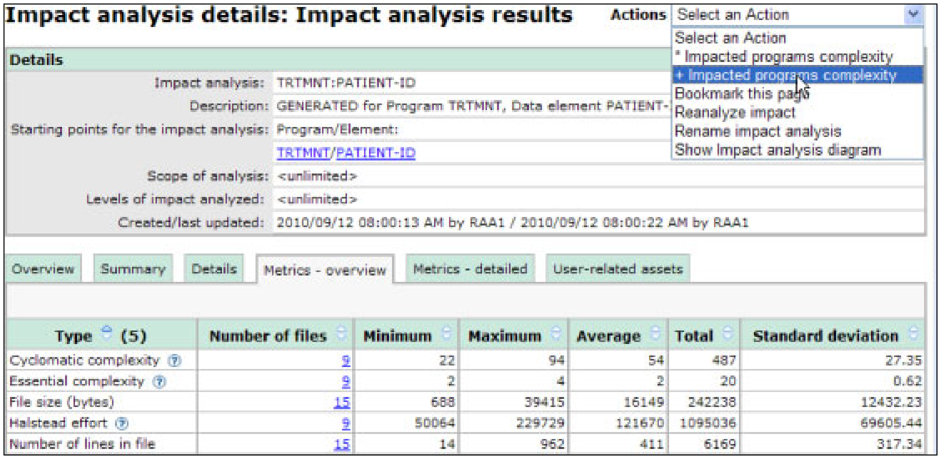


Figure 5-66 Impact analysis details: Metrics overview tab showing complexity statistics for the application elements impacted

Figure 5-67 shows a custom query that breaks the Impact Analysis costing down to the level of individual programs. This report is possibly the most useful of all the metrics data points.

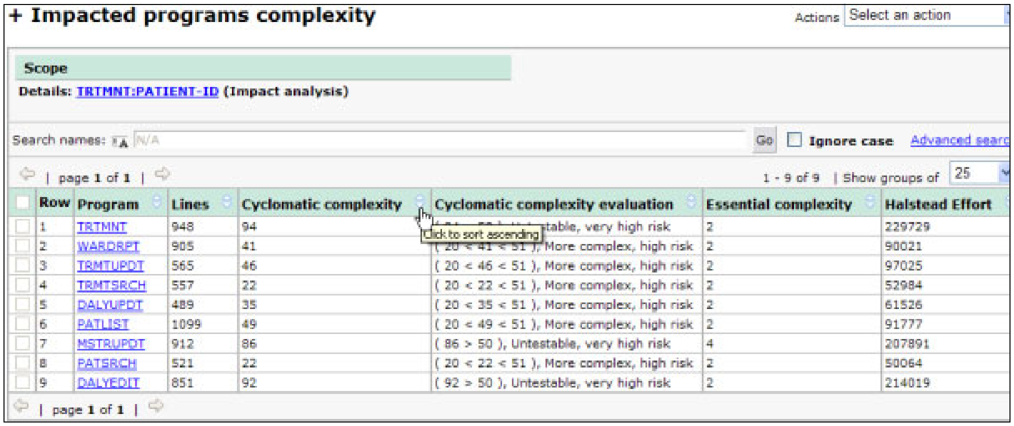


Figure 5-67 A Rational Asset Analyzer custom query that breaks out the complexity of each program to be changed

5.11.6 Reexamining a saved Impact Analysis report

Follow these steps to access a saved Impact Analysis report:

1. From any of the Rational Asset Analyzer web pages, you will see an Impact Analysis menu. When you move the mouse pointer over the menu, it offers a submenu with View all or View mine options. Either choice opens a Summary (list) page, as shown in Figure 5-68. By selecting View mine, you see only the Impact Analysis reports that you generated.
2. To reexamine an Impact Analysis report, click the hyperlink.

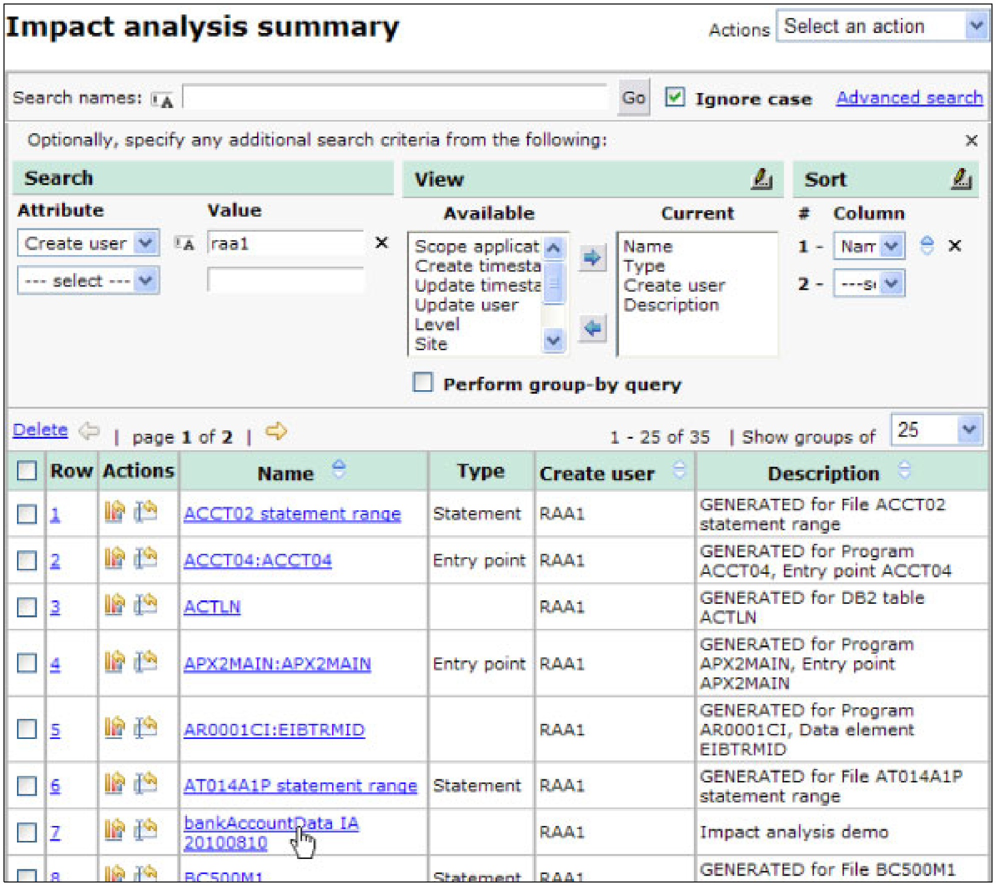


Figure 5-68 The saved Impact Analysis reports: Available for re-examination

5.12 Custom queries

You have already been exposed to several custom queries. A custom query is an SQL statement that accesses information in the Rational Asset Analyzer repository and displays it through a standard Rational Asset Analyzer web page.

Because the repository is a DB2 database, you can create your own SQL queries to perform these tasks:

- Analyze information in ways that are not built into the predefined Rational Asset Analyzer analysis pages
- Extend the repository information by defining, loading, and querying your own custom tables and by combining their metadata with Rational Asset Analyzer's metadata.

  For example, you might wish to define a job scheduling table to document the sequence with which batch jobs are scheduled and launched. You can use standard SQL Data Definition Language (DDL) to create the table, using the IBM Rational Developer for System z tools to populate the table with scheduling information. You then can write custom queries that join the batch scheduling table to the batch job metadata tables to produce custom job reports that include scheduling semantics.

5.12.1 Custom query library or collection of sample custom queries

Custom queries are extremely useful; therefore, the Rational Asset Analyzer development team has created and maintains an importable selection of them (Figure 5-69). This set of custom queries is available as an XML file that is added to the Rational Asset Analyzer default installation directory:

- Typically: `<`*`Rational Asset Analyzer installation folder`*`>\sample\CQ`

  `C:\Program Files\IBM\Rational Asset Analyzer\sample\CQ`
- In this directory, you find an XML file with the name:

  `Custom Query Samples +Collection_2009.01.06_CqExport.xml`
- In this directory, you will find a PDF file with the name of `RAA Custom Query Samples +Collection.pdf`

The custom queries in this collection are a mix of these types:

- Administrative:
  - Show duplicate source files in a container
  - Data sets with no references
  - The ability to edit and copy the custom query samples
- Management reporting:

  Complexity metrics for individual programs and programs impacted by a change
- Analytic:
  - DB2 tables accessed by a job's application programs
  - IMS database segments accessed by a job's programs
  - BMS maps used in a transaction

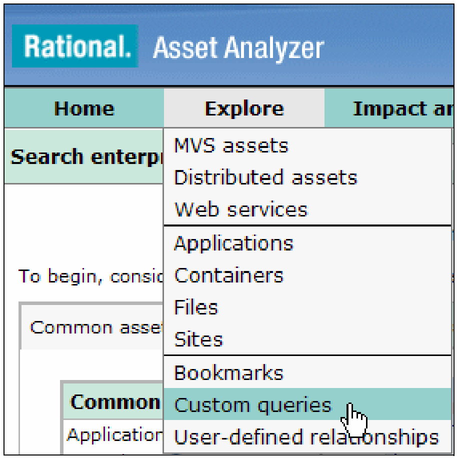


Figure 5-69 Access to the custom queries functionality

Follow these steps to import the samples:

1. From the Rational Asset Analyzer Home page (Figure 5-69 on page 292), click **Explore** → **Custom queries**.
2. This action displays the Custom queries page (Figure 5-70). There is an Actions option to import an exported set of queries. We have imported an exported set of queries with the Rational Asset Analyzer-shipped samples. Other options from the Custom queries page are available to maintain your queries, export them (for other users to use), run them, and so forth.

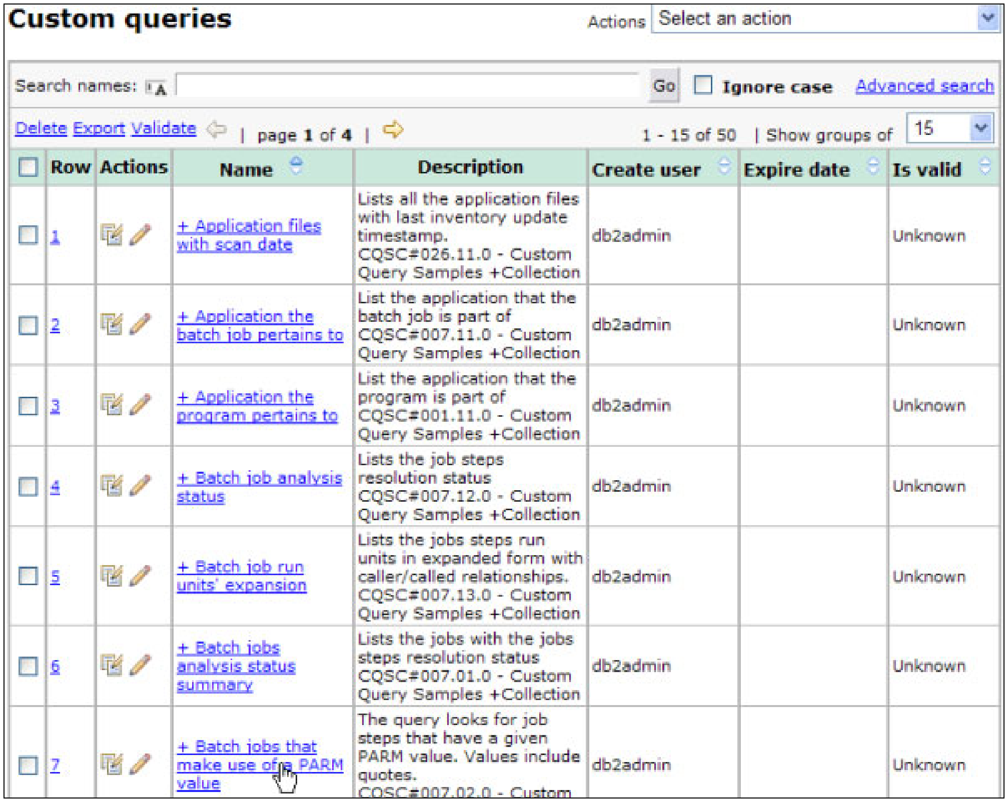


Figure 5-70 The Custom queries detailed page for creating and managing your queries

3. You can also create a new custom query from a wizard that is launched in the Actions drop-down on the Custom queries page (Figure 5-71).

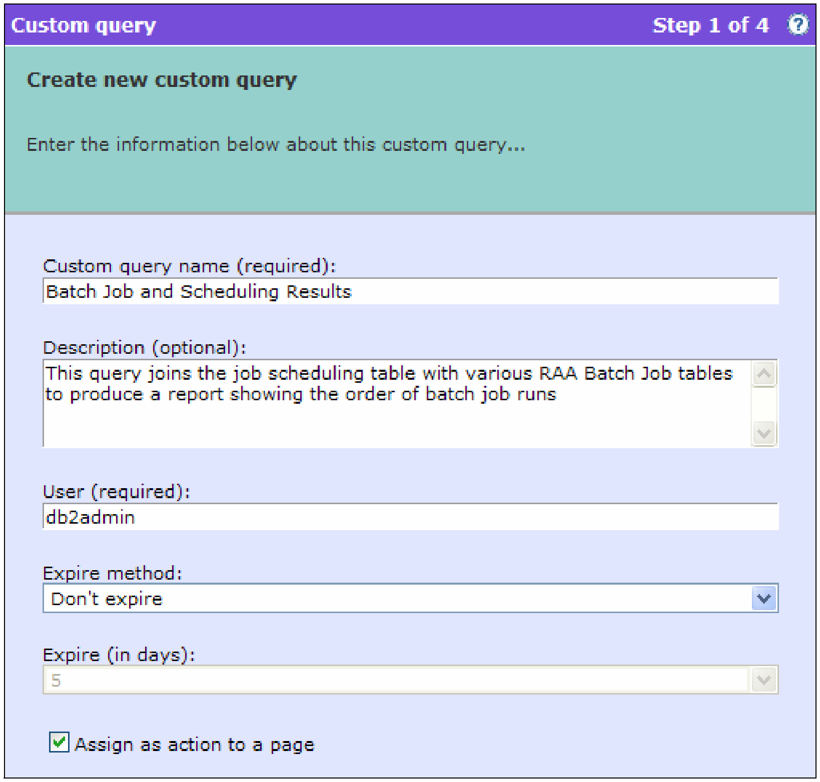


Figure 5-71 Wizard (step 1 of 4) for creating a new custom query

4. In Figure 5-71, you begin the process of creating a new custom query by defining a name, optionally describing the query, specifying the retention period for the query, and choosing whether to assign the custom query to an existing Rational Asset Analyzer analysis web page as an action (Figure 5-72).

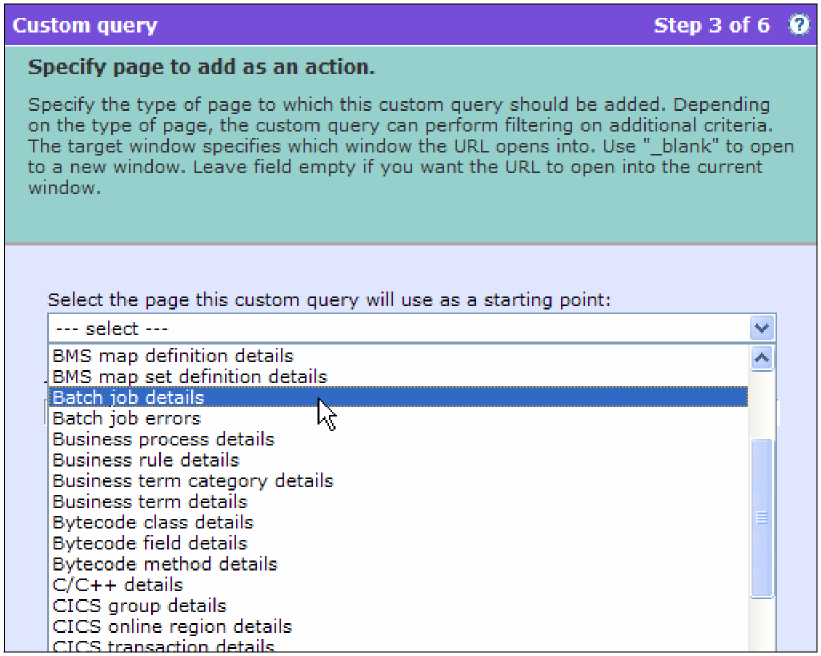


Figure 5-72 Selecting an existing Rational Asset Analyzer web page Actions menu for your custom query

5. Figure 5-73 shows the wizard's step for entering the SQL statement that runs as the custom query. The descriptive comments explain the taxonomy of the statement variables. However, note that a high level of SQL expertise is required. The Rational Asset Analyzer repository model is normalized so that three, four, and five table joins are common. Also, an understanding of the parent-child table relationships is necessary. This information, which we describe in 5.10.1, "Dependency diagrams" on page 268, is well documented.

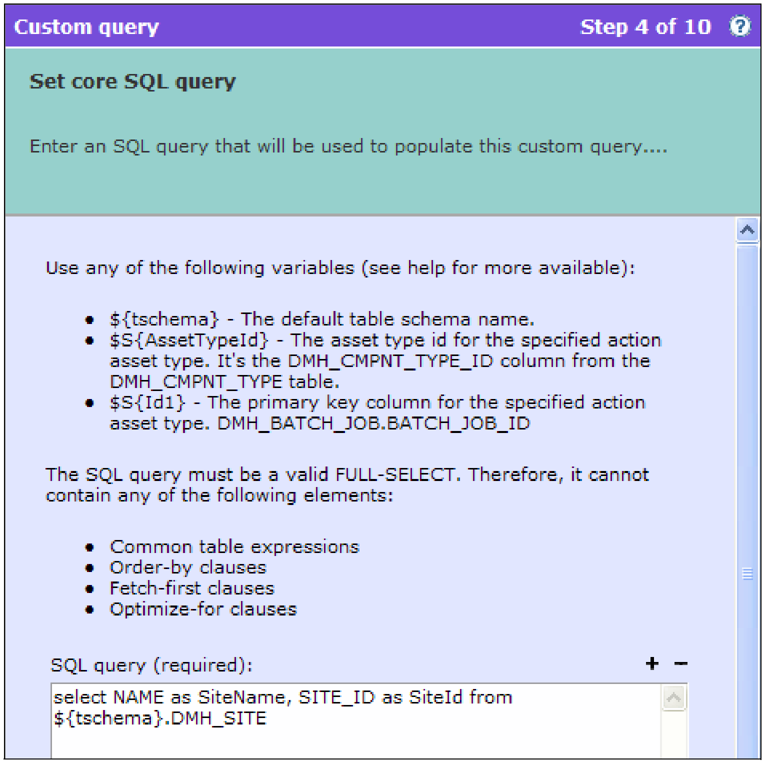


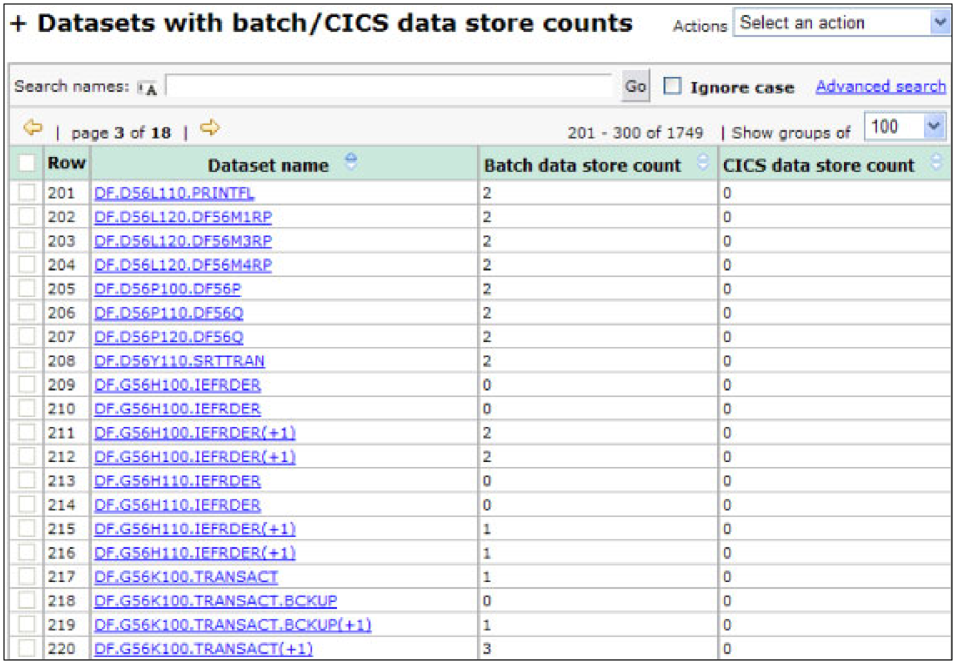
Figure 5-73 The custom query wizard step where you code your SQL query

5.12.2 Sample custom queries

Here we show two more examples of custom queries.

The first example of custom query is the Data sets with batch/CICS data store counts custom query that is shown in Figure 5-74. This custom query lists data sets with the counts of existing entries in `DD_CONCAT` and the entries in `ONLINE_DSTORE`. Use this custom query to find data sets that are used in batch and defined in CICS.

To generate this report, after importing the custom queries that are shipped with IBM Rational Developer for System z from any Data sets Summary page, open the Actions drop-down menu and select **+ Data sets with batch/CICS data store counts**.

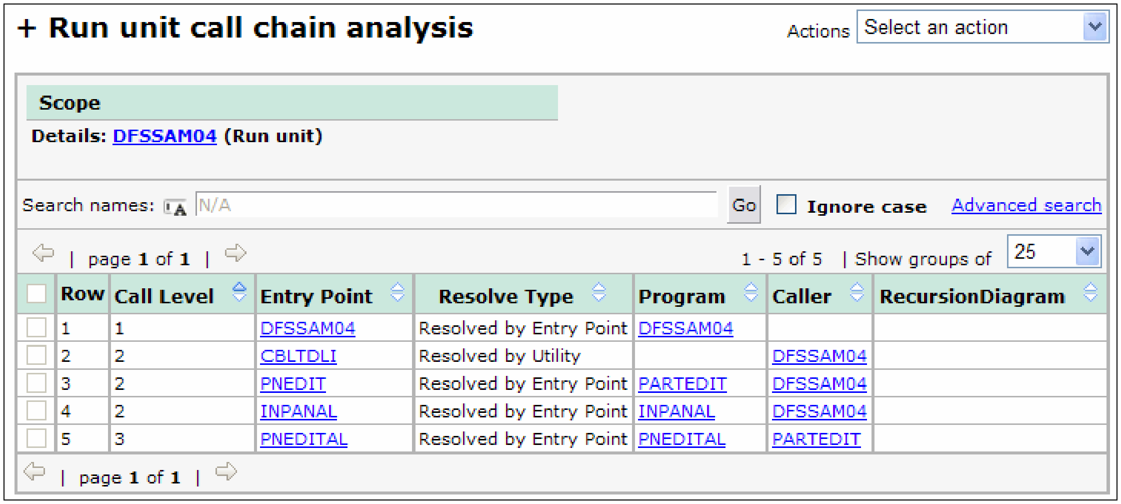


Row	Dataset name	Batch data store count	CICS data store count
201	DF.D56L110.PRINTFL	2	0
202	DF.D56L120.DF56M1RP	2	0
203	DF.D56L120.DF56M3RP	2	0
204	DF.D56L120.DF56M4RP	2	0
205	DF.D56P100.DF56P	2	0
206	DF.D56P110.DF56Q	2	0
207	DF.D56P120.DF56Q	2	0
208	DF.D56Y110.SRTTRAN	2	0
209	DF.G56H100.IEFRDER	0	0
210	DF.G56H100.IEFRDER	0	0
211	DF.G56H100.IEFRDER(+1)	2	0
212	DF.G56H100.IEFRDER(+1)	2	0
213	DF.G56H110.IEFRDER	0	0
214	DF.G56H110.IEFRDER	0	0
215	DF.G56H110.IEFRDER(+1)	1	0
216	DF.G56H110.IEFRDER(+1)	1	0
217	DF.G56K100.TRANSACT	1	0
218	DF.G56K100.TRANSACT.BCKUP	0	0
219	DF.G56K100.TRANSACT.BCKUP(+1)	1	0
220	DF.G56K100.TRANSACT(+1)	3	0

Figure 5-74 Custom query that displays data sets that are used both in batch jobs and as CICS data stores

The second example is the Run unit call chain analysis custom query that is shown in Figure 5-75. This custom query reveals the program/subprogram called/calling module chain.

To generate this report, after importing the custom queries shipped with IBM Rational Developer for System z from any Run Unit summary or details page, open the Actions drop-down menu and select **+ Run unit call chain analysis**.



Row	Call Level	Entry Point	Resolve Type	Program	Caller	RecursionDiagram
1	1	DFSSAM04	Resolved by Entry Point	DFSSAM04		
2	2	CBLTDLI	Resolved by Utility		DFSSAM04	
3	2	PNEDIT	Resolved by Entry Point	PARTEDIT	DFSSAM04	
4	2	INPANAL	Resolved by Entry Point	INPANAL	DFSSAM04	
5	3	PNEDITAL	Resolved by Entry Point	PNEDITAL	PARTEDIT	

Figure 5-75 A stored custom query perfoms detailed called/calling chain analysis within a run unit

5.13 Application documentation with Rational Asset Analyzer

Of course, Rational Asset Analyzer's most obvious value is in its application analysis functionality. However, most of the Rational Asset Analyzer clients derive technological advantage by re-purposing many of the reports that we have discussed by integrating them into their system documentation.

There are numerous methods to integrate the reports into your system documentation:

- All of the diagrams can be saved separately (Figure 5-76 on page 299).
- You can copy and paste all of the diagrams, typically as bitmaps, unless you use Adobe® technology (Figure 5-77 on page 300).
- You can select, copy, and paste any of the web pages into spreadsheets and other documentation.

5.14 Hyperlinked documentation

With both the saved Scalable Vector Graphics (SVG) diagrams (Figure 5-76) and with saved Rational Asset Analyzer web pages where you have copied and pasted rows with hyperlinks, those URLs are active or *"live"* and make the documentation dynamic. By clicking a link, Rational Asset Analyzer opens to the asset that the link exposed, which is typically a specific element in the metadata repository.

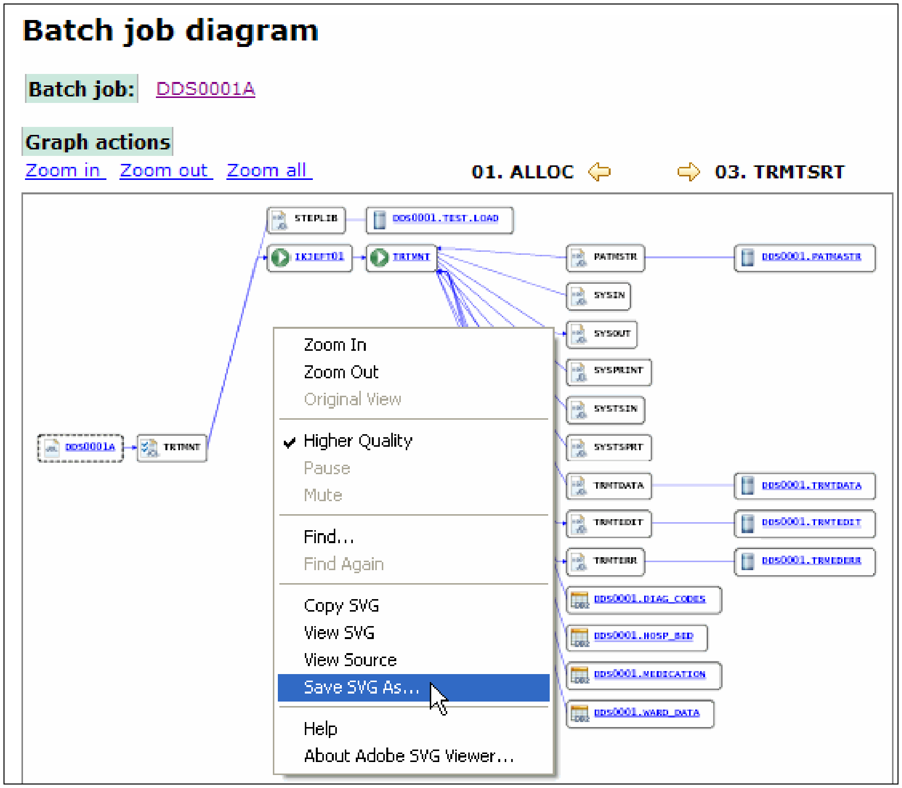


Figure 5-76 Saving a diagram as an SVG file

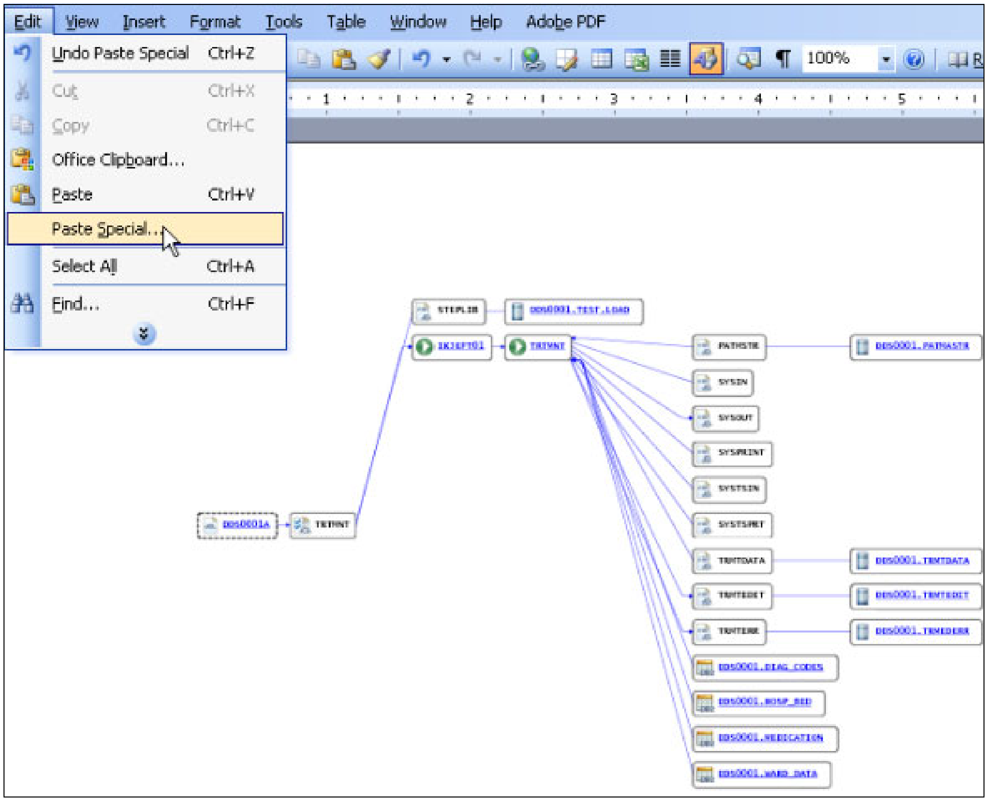


Figure 5-77 Pasting an SVG diagram into a document as a bitmap

5.15 Summary

We have described these topics:

- What Rational Asset Analyzer is and how it performs static code analysis
- The process of loading inventory into the Rational Asset Analyzer repository
- The use of Rational Asset Analyzer by various stakeholders
- The primary use cases for Rational Asset Analyzer:
  - Project management
  - Application analysis
  - Application understanding
  - Impact analysis
  - Application documentation
- Extending Rational Asset Analyzer through custom queries

6

IBM Problem Determination Tools

IBM Problem Determination Tools for z/OS (PD Tools) consist of a core group of IBM products that are designed to work in conjunction with compilers and run times to provide a start-to-finish development solution for the IT professional. This book provides an introduction to the tools, guidance for program preparation for use with them, an overview of their integration, and several scenarios for their use.

When developing a system, developers typically use an iterative process. IBM File Manager for z/OS can scramble, alter, or otherwise process this data. After you create the data, you can use this data to test the newly developed or maintained software.

If an ABEND during testing occurs, IBM Fault Analyzer for z/OS enables you to quickly and easily pinpoint the ABENDing location and, optionally, the failing line of code. Many times, this location is sufficient for you to correct the problem. But, occasionally, it might be necessary to delve a little deeper into the code to figure out the problem. IBM Debug Tool for z/OS allows you to step through the code at whatever level is required to determine where the error was introduced or encountered.

This chapter contains the following tool topics:

- Debug Tool for z/OS and Debug Tool Utilities and Advanced Functions
- Fault Analyzer for z/OS
- File Manager for z/OS

These tools provide you with a comprehensive suite of products that can help improve application delivery and enhance production availability and performance.

6.1 Introduction to PD Tools

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

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

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

6.2 Installing the PD tools on System z

You install all the PD tools (Fault Analyzer, File Manager, and Debug Tool) using SMP/E for z/OS. In this section, we describe the SMP/E installation and we list, at a high level, the steps that you need to execute to install these products. You can obtain more details about exactly what you need to do for each product on the product library page, in the program directory PDF.

6.2.1 Installation requirements and considerations

The following sections identify the system requirements for installing and activating File Manager, Fault Analyzer, Debug Tool and IBM Application Performance Analyzer for z/OS. We use the following terminology:

- Driving system: The system that is used to install the program and where SMP/E executes.

  The program might have specific operating system or product-level requirements for using processes, such as binder or assembly utilities during the installation.

- Target system: The system on which the program is configured and run.

  The program might have specific product-level requirements, such as needing access to the library of another product for link-edits. These requirements, either mandatory or optional, might directly affect the product during the installation or in its simple or enhanced operation.

In many cases, you can use a system as both a driving system and a target system. However, you can make a separate clone that you can IPL of the running system to use as a target system. The clone must include copies of all the system libraries that SMP/E updates, copies of the SMP/E consolidated software inventory (CSI) data sets that describe the system libraries, and your PARMLIB and PROCLIB.

Use separate driving and target systems in the following situations:

- When you install a new level of a product that is already installed, the new level of the product replaces the old level. By installing the new level onto a separate target system, you can test the new level and keep the old level in production at the same time.
- When you install a product that shares libraries or load modules with other products, the installation can disrupt the other products. By installing the product onto a separate target system, you can access these effects without disrupting your production system.

6.2.2 SMP/E considerations for installation

Use the SMP/E RECEIVE, APPLY, and ACCEPT commands to install this release.

6.2.3 SMP/E options subentry values

Table 6-1 shows the suggested values for certain SMP/E CSI subentries. Using values lower than these can result in failures in the installation. DSSPACE is a subentry in the GLOBAL options entry. PEMAX is a subentry of the GENERAL entry in the GLOBAL options entry. See the SMP/E manuals for instructions to update the global zone.

Table 6-1 Suggested values for certain SMP/E CSI subentries

Subentry	Value	Comment
DSSPACE	300,150, 250	Space allocation for SMPTLIB data sets.
PEMAX	SMP/E default	We advise using the SMP/E default for PEMAX.

6.2.4 Allocating and initializing the SMP/E data sets (optional)

You can install these products in the same SMP/E zone as z/OS V1.9.0 (or later), or in a separate zone:

- If you install into existing SMP/E data sets, ensure that you have enough space.
- If you plan to install into an existing zone, you must have already allocated and primed the cluster.
- You can go on to the next step to perform an SMP/E RECEIVE.
- To install into a new zone, edit and submit the sample job for the particular product to define and prime a new SMP/E CSI cluster. Consult the instructions in the sample job for more information.

See each product's documentation for more details:

- Debug Tool for z/OS:

  `http://www-01.ibm.com/software/awdtools/debugtool/library/`
- Application Performance Analyzer for z/OS:

  `http://www-01.ibm.com/software/awdtools/apa/library/`
- File Manager for z/OS:

  `http://www-01.ibm.com/software/awdtools/filemanager/library/#v101`
- Fault Analyzer for z/OS:

  `http://www-01.ibm.com/software/awdtools/faultanalyzer/library/`

Now, follow these steps to install each product:

1. Perform SMP/E RECEIVE.
2. Allocate SMP/E target and distribution libraries.
3. Allocate file system paths.
4. Create DDDEF entries.
5. Perform SMP/E APPLY.
6. Perform post-installation tasks for the Application Performance Analyzer.
7. Run the installation verification program (IVP).
8. Perform SMP/E ACCEPT.

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

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

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

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

6.4 Using Debug Tool

The IBM Debug Tool can interface with Rational Developer for System z to provide an interactive source debugging of z/OS applications. Debug Tool helps you debug applications running in a variety of environments, such as batch, CICS, IMS, DB2 stored procedures, and UNIX System Services.

In this section, we introduce the integration of Debug Tool with the Rational Developer for System z interface and explain how this integration can enhance the process of developing applications for System z. Debug Tool consists of complementary products, which enable symbolic and source code debugging of COBOL, PL/I, C, C++, and IBM High Level Assembler for System z (HLASM) code. The symbolic information that is made available in the debugging perspective of Rational Developer for System z includes entry point names, variable names, labels, and registers including general purpose registers, as well as floating point registers.

The debugger allows you to control the execution of your program by setting breakpoints, suspending launched programs, stepping through your code, and examining the contents of variables. The Rational Developer debugger has a client/server design so that you can debug programs that run remotely on other systems in the network, as well as debug programs that run locally on your workstation. The *debug client* runs inside the Rational Developer workbench on your workstation. The *debugger server* runs on the same system as the program that you want to debug, which can be a program launched on your workstation (local debugging) or a program started on a computer that is accessible through a network (remote debugging).

6.4.1 Debugging COBOL, PL/I, Assembler, and C++ programs

The most dramatic difference between Debug on the 3270 interface and Debug on Rational Developer is the graphical user interface (GUI). The 3270 interface provides access to a super-set of the Debug Tool's functions, but Rational Developer provides a GUI front end to a large number of Debug's capabilities.

To use an interactive debugger, we must inform the debugger where to interact with us (the user). We can use an IBM VTAM® 3270 interface (also known as a green screen), but alternatively we can route the interactive debug session to Rational Developer. To do this, in the Rational Developer debug perspective, ensure that Rational Developer is listening for an interactive debug session (Figure 6-1) and determine the IP address of our Rational Developer client. We then direct the debug session to the IP address of our Rational Developer client.

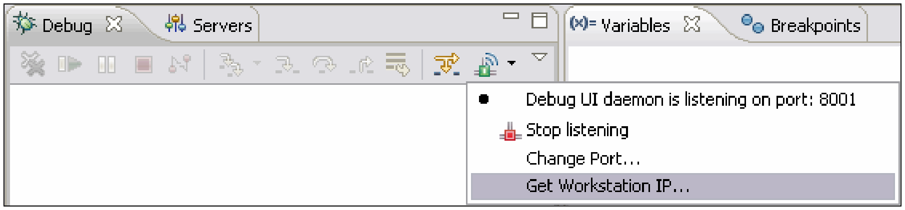


Figure 6-1 Listening for the debug session

To switch to the debug perspective, use the Window pull-down (**Window** → **Open Perspective** → **Other**) in the Open Perspective pop-up and select **Debug**.

When working with Rational Developer, you can use a z/OS-based debugging engine to debug code in an MVS subproject, or you can use a workstation-based debugging engine to debug code in a local project. For z/OS-based debugging, you must be in the online state. When working on a local project, you can work remotely. The user's experience for both z/OS-based and local debugging is similar, and you can execute the following tasks on the source code during your debug efforts.

After launching a debug session, there are debug views available that provide access to a variety of debug tasks. Table 6-2 lists the views that are available for debugging.

Table 6-2 Default views for the Debug perspective

View	Description
Debug view	Allows you to manage program debugging
Debugger editor	Shows source for your program
Breakpoints view	Offers a convenient location for setting and working with breakpoints
Variables view	Contains a list of all variables in your application and allows you to edit variables
Registers view	Shows registers in your program
Monitors view	Provides a convenient location for working with variables, expressions, and registers that you choose to monitor
Modules view	Shows a list of modules loaded while running your program
Debug Console	Allows you to issue commands to the debug engine, view output from the engine, and see results of commands that you have issued
Memory view	Allows you to view and map memory used by your application

Call stack

A *call stack* is a dynamic stack data structure that stores information about the active subroutines of a computer program. You can use a call stack for several related purposes, but the key reason for a call stack is to keep track of the point to which each active subroutine must return control when it finishes executing. With Rational Developer, Figure 6-2 shows a call stack when a program is suspended.

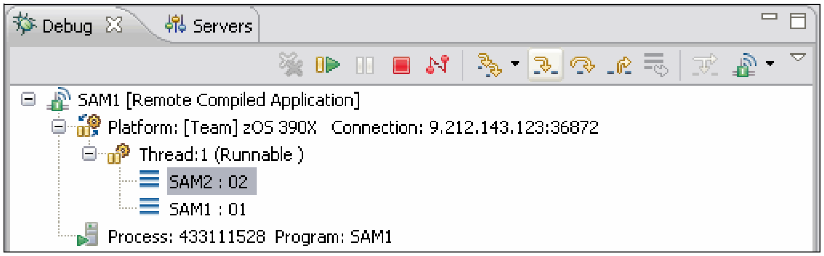


Figure 6-2 Call stack

Setting and learning breakpoints at a specific line

Breakpoints are temporary markers that you place in your executable program to tell the debugger to stop your program at a given point. When a breakpoint is encountered, execution suspends at the breakpoint before the statement is executed. At this point, you can see the stack for the thread and check the contents of variables, registers, and memory. You can then *step over* (execute) the statement and see what effect it has on the argument.

To access the wizards for setting breakpoints, right-click in the Breakpoints view and select **Add Breakpoint** from the pop-up menu. This selection expands to a menu that allows you to choose the breakpoint type that you want to set. When you use the wizard to set a breakpoint, you can specify optional breakpoint parameters and set conditional breakpoints. See Figure 6-3.

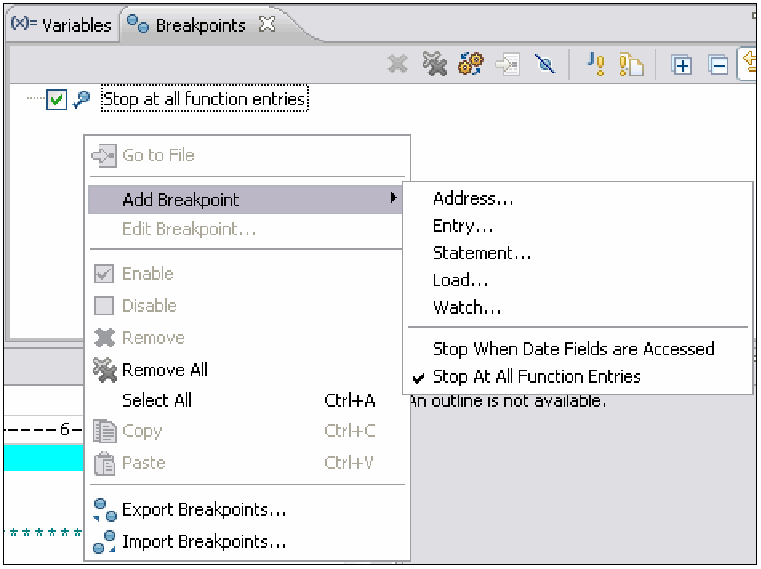


Figure 6-3 Adding a breakpoint

Table 6-3 shows the breakpoint types that are supported when debugging System z applications.

Table 6-3 Breakpoint descriptions

Type	Description
Statement breakpoints	Triggered when the statement they are set on is about to be executed
Entry breakpoints	Triggered when the entry points they apply to are entered
Address breakpoints	Triggered before the disassembly instruction at a particular address is executed
Watch breakpoints	Triggered when execution changes data at a specific address
Load breakpoints	Triggered when a dynamic link library (DLL) or object module is loaded
Exception breakpoints	Triggered when an exception that is recognized by the debugger is thrown

Exception breakpoints are set in the Breakpoints view by clicking **Manage Compiled Language Exception breakpoints** and then by selecting the exception type that you want the debugger to catch in the Manage Exception Breakpoints dialog box. The exception types that are available when debugging System z are TEST(ALL), TEST(ERROR), and TEST(NONE). You can select only one of these selection types.

You can set statement breakpoints in the Debugger editor by double-clicking the ruler area to the left of a statement, by a right-click pop-up menu action, or by using the wizard in the Breakpoints view.

You can set entry breakpoints in the Modules view by right-clicking an entry point and selecting **Set entry breakpoint** from the pop-up menu or by using the wizard in the Breakpoints view (Figure 6-4).

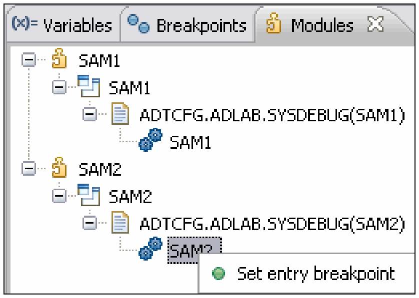


Figure 6-4 Setting an entry breakpoint

In addition, you can right-click the debug target (or one of its threads or stack frames) in the Debug view and select **Options** → **Stop At All Function Entries** from the pop-up menu to stop at all entry points (this option is also available in the Breakpoints view pop-up menu). You set all other breakpoint types by wizard in the Breakpoints view.

The Breakpoints view shows a list of all breakpoints (for all debug sessions), unless you use the filter by debug target action or link the Breakpoints view to the Debug view. To filter out breakpoints that are not related to the current debug session, click **Show Breakpoints Supported by Selected Target** in the Breakpoints view. To link the Breakpoints view with the Debug view, click **Link with Debug View**. When this toggle is selected and a breakpoint suspends a debug session, that breakpoint is automatically selected in the Breakpoints view.

The breakpoint entries in the list provide you with a summary of the breakpoints' properties, in brackets. With pop-up menu options, you can add breakpoints, remove breakpoints, and enable or disable breakpoints. You can also edit breakpoint properties with a pop-up menu option. With push buttons in the Breakpoints view, you can remove breakpoints.

You can enable and disable Breakpoints with pop-up menus in the Breakpoints view or the editor and by check box in the Breakpoints view. When a breakpoint is enabled, it causes all threads to suspend whenever it is hit. When a breakpoint is disabled, it does not cause threads to suspend.

In the Breakpoints view, two indicators show to the left of a set breakpoint. To the far left is a check box, which indicates if the breakpoint is enabled (when enabled, the check box contains a check mark). To the near left, an indicator with a check mark overlay indicates a breakpoint that has been successfully installed by the debug engine (if the breakpoint is enabled, this indicator is filled; if the breakpoint is disabled, this indicator is not filled). In the editor, statement breakpoints are indicated by an indicator with a check mark overlay, indicating a breakpoint that has been successfully installed by the debug engine (if the breakpoint is enabled, this indicator is filled; if the breakpoint is disabled, this indicator is not filled).

You must install breakpoints before they will suspend execution. It is possible to add a breakpoint that is invalid for the current debug session. This breakpoint is not installed until it is part of a debug session that includes a debug engine that will recognize the breakpoint.

In the editor, statement and entry breakpoint indicators show in the marker bar to the left of the editor. Indicators for statement, entry, address, watch, and load breakpoints show in the Breakpoints view.

While in the Breakpoints view, the source editor opens to the location of a breakpoint if you perform one of the following actions:

- Double-click the breakpoint.
- Select the breakpoint and click **Go to File For Breakpoint**.
- Right-click the breakpoint and select **Go to File** from the pop-up menu.

Using step procedures

After the code is suspended, you are able to use the stepping procedures, which allow you to step through the execution of the program line-by-line using the step controls. While performing a step operation, if a breakpoint or exception is encountered, execution suspends at the breakpoint or exception, and the step operation ends. You can use step commands to step through your program a single instruction or location at a time. Table 6-4 lists the available step commands.

Table 6-4 Step commands

Command	Description
Step Over	Called functions run without stepping into it.
Step Into	Program runs to the next hook or statement.
Animated Step Into	The debugger issues a Step Into action repeatedly. You can control the delay between each step by selecting the Animated Step Into icon down arrow.
Step Return	Program runs to the return point (just after the call point).

Select the Step Into command to step into the next statement at the currently executing line of code. As you step through the program, the line of code which you are currently executing is highlighted, as shown in Figure 6-5. The values of the variables that are currently being used show and change as the code executes.

```
           ACCEPT CURRENT-TIME FROM TIME.
           DISPLAY 'SAM1 STARTED DATE = ' CURRENT-MONTH '/'
                  CURRENT-DAY '/' CURRENT-YEAR '   (mm/dd/yy)'.
           DISPLAY '             TIME = ' CURRENT-HOUR ':'
                  CURRENT-MINUTE ':' CURRENT-SECOND.
```

Figure 6-5 Current line of execution

Select the Step Over command to step over the next method call (without entering it) at the currently executing line of code. Even though the method is skipped as far as stepping is concerned, it still executes.

Using variable tools

Any function or program executes to change data in an expected way, so a key part of debugging is to locate and determine if the program is acting on the data correctly. The first step is finding a variable. Locating a variable and determining its value in a complex program while debugging can be tedious. Rational Developer and Debug have solutions for this challenge. In one method, when a breakpoint is set and the program is suspended, you can use the *hover* feature. This feature allows you to see a variable's current value by placing the mouse pointer over the variable name while the program is suspended (Figure 6-6).

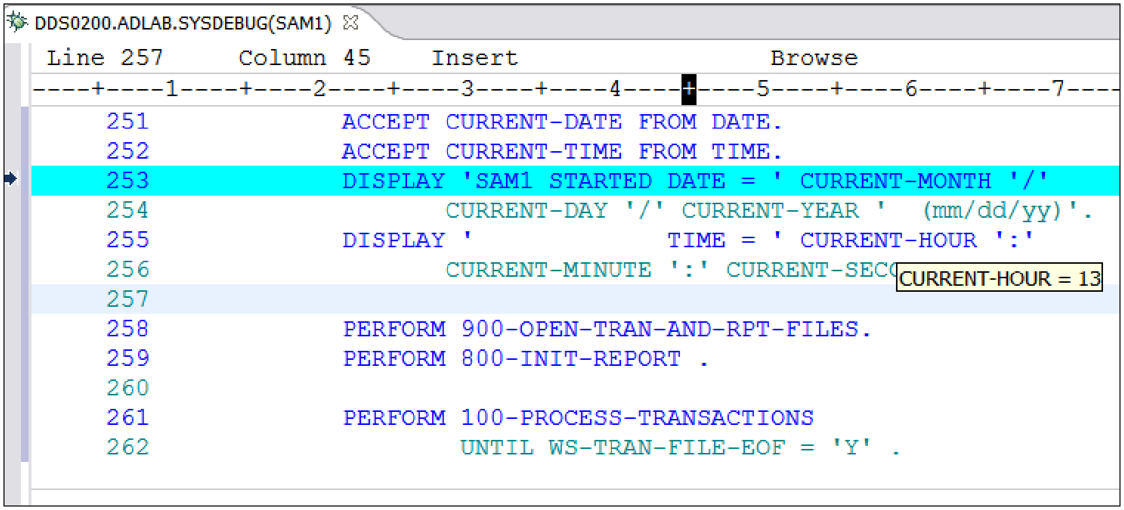


Figure 6-6 Variable hover feature

You can also set up a Variables window that will show the variable name and its value as you step through the program. To open the Variables window, go to Window in the top tool bar, select **Show View**. Then, near the bottom of the window, you see Variables. Or, you can press Alt+Shift+Q and then press V, and a window pops up near your Breakpoints window, looking similar to Figure 6-7.

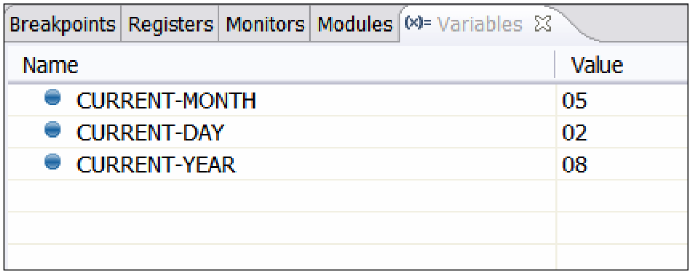


Figure 6-7 Current variable value display

In the Variables view, you can dynamically change the variable values as you step through the code, which allows you to enter the correct data into the variable for testing the rest of the program.

6.4.2 Right-clicking in the code space

During a Debug session, you right-click to access many features. Right-clicking in the code space during a debug session gives access to functionality that is useful in debugging source code. Table 6-5 shows several right-click actions and their descriptions.

Table 6-5 Right-click menu in the code space

Action name	Description	Hot keys
Find Text	Allows you to enter a string of characters to find within the code.	Ctrl+F
Find Function or Entry Point	Allows you to enter the name of the function or entry point for which you are searching.	Ctrl+F10
Add Watch Breakpoint	Allows you to set up a watch breakpoint with the wizard that pops up. Set up the number of bytes to watch and the frequency.	N/A
Monitor Expression	Enables you to select the thread that has the expression that you want to monitor, and the tool places a monitor in the Monitors tab.	N/A
Monitor Memory	Allows you to add a new memory monitor, which lets you view and change the contents of memory or memory areas that are used by your program.	N/A
Run To Location	Executes the program to the currently selected line in the editor.	N/A
Jump To Location	Jumps to the currently selected line in the editor without execution.	N/A
Edit Source Lookup Path	Allows you to edit the path that is used to locate source files.	N/A
Change Text Files	Allows you to enter the name of an overriding file for the source.	N/A
Switch Views	Enables you to change views.	N/A

6.5 IBM Fault Analyzer for z/OS

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

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

Fault Analyzer records each application ABEND in a fault-history file by job name, failure code, and other details, along with the analysis report and storage pages that are referenced during the analysis. You can retrieve this information later to reanalyze the failure.

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

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

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

6.6 Using Fault Analyzer

The Fault Analyzer plug-in for Rational Developer for System z helps you to simplify ABEND analysis by connecting to a remote IBM z/OS system and provides access to problem reports for diagnosing mainframe application errors and ABENDs. The Fault Analyzer plug-in for Rational Developer for System z has these key features:

- An interface to manage views and multiple fault history files
- The ability to browse fault entries that were created during real-time analysis of ABENDing programs
- A browser for browsing the dump storage that is associated with a fault entry
- A source listing of ABENDing programs using side files

Fault Analyzer Integration allows users to perform the following tasks:

- Browse Fault Analyzer ABEND reports on CICS, IMS, batch, Java, WebSphere, and other runtime environments.
- View dump selections relating to ABENDs.
- Annotate reports to share comments with other users who browse the same reports.
- View a new split-screen mode in the dump analyzer, allowing users to view disparate parts of the same dump contents to compare data in separate parts of the dump easily.
- Use includes that reference the same editor sessions that are used for code editing instead of module side files. This capability offers a more integrated experience using the same data for all PD Tool integrations and developer sessions leading from editing to fault analysis more seamlessly.

6.6.1 Fault Analyzer Artefacts view

The Fault Analyzer Artefacts (FA Artefacts) view (Figure 6-8) enables you to register a set of history files that can be browsed to identify the cause of failures in your applications. Also, this view allows you to monitor the contents of history files to alert you when a failure occurs in one of your applications.

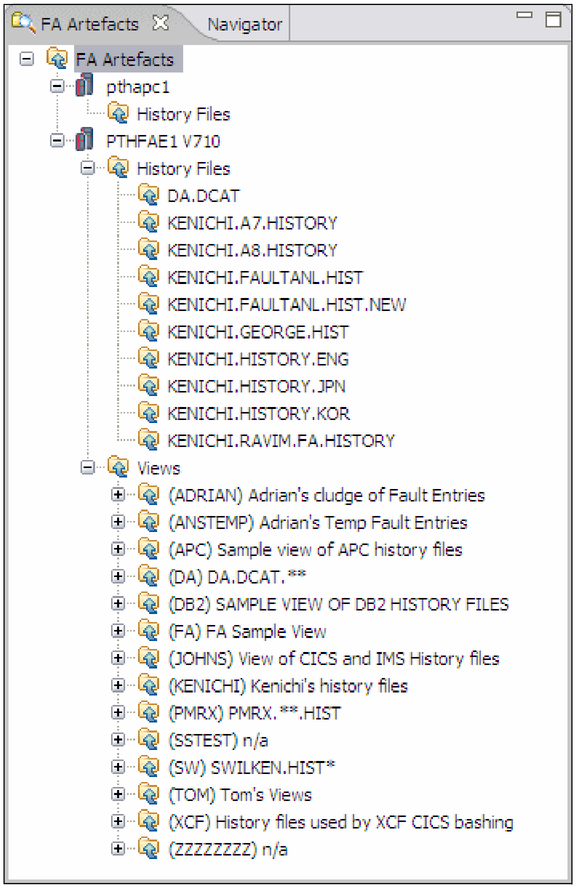


Figure 6-8 FA Artefacts view

History files are organized in a tree structure in which the root of the tree is the Fault Analyzer artefacts. The artefact contains children that represent separate mainframe systems. Each system node potentially contains two children: one child for containing all history files for the system and another child for containing the view information. When you double-click a history file element, all fault entries that are contained in the selected history file show in the Default view. A *view* is a Fault Analyzer concept that enables you to group a set of history files. When you select a view element, fault entries that are contained in all the history files that are defined in the selected view show in the Default view.

You can define views on the host and retrieve views using the Retrieve View Information action from the context menu in the FA Artefacts view. You can set up column configurations in both Rational Developer for System z and the host; however, Rational Developer for System z column definitions are not saved to the host. See the “Setting up views” section in the *Fault Analyzer for z/OS V8R1 User’s Guide and Reference*, SC19-1253-05.

To specify a history file in the FA Artefacts view, right-click **FA Artefacts** and select **Add New History File**. To populate the History File Default view, right-click the Fault Analyzer history file name and select **Populate/Refresh History File**.

Invoking the Fault Analyzer ISPF interface stores the history files that you are using as part of your ISPF profile information. Follow these steps to retrieve the information of your last-accessed history files:

1. In the FA Artefacts view, right-click **FA Artefacts** and select **Retrieve Last Access History Files**.
2. When the Retrieve ISPF Profile window opens, use the arrow for the drop-down menu to select the following information for these fields (Figure 6-9):
   - Existing RSE Connection: Select a connection.
   - ISPF Profile Name: Select the name of your ISPPROF data set.
   - Temporary DSN: Choose a name for a temporary ISPPROF data set.
   - Exec Name: Select the Fault Analyzer SIDIEXEC(IDICSXDS) data set.
   - FA Application ID: Select the IDI.

See Figure 6-9 for an example of information retrieval.

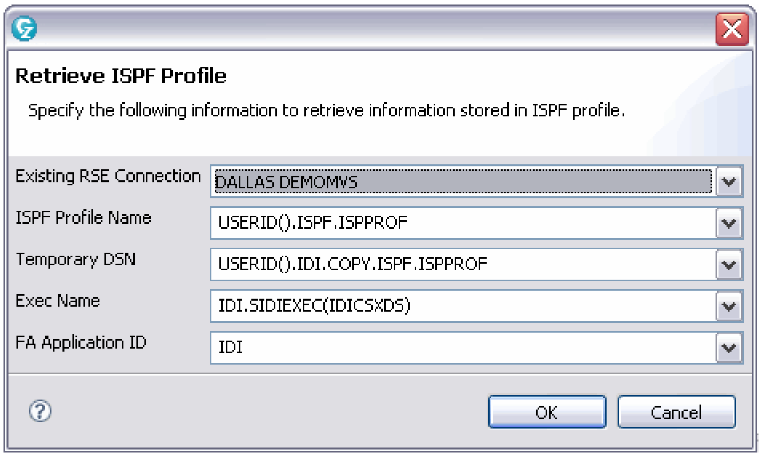


Figure 6-9 Retrieve ISPF Profile window

Table 6-6 describes these fields.

Table 6-6 Retrieve ISPF Profile field descriptions

Field names	Description
Existing RSE Connection	Name of the Remote Systems Explorer connection profile. The ISPF profile information is retrieved from the specified host.
ISPF Profile Name	Name of a data set that contains your ISPF profile.
Temporary DSN	Name of a temporary data set that is used to copy your ISPF profile. You must have permission to allocate and write to the data set.
Exec Name	Data set name that contains the Restructured Extended Executor (REXX) or exec to retrieve your ISPF profile information. The default location is IDI.SIDIEXEC(IDICSXDS). Customization is required after the host installation (refer to the exec for further information).
FA Application ID	Application ID of Fault Analyzer installation on the host. The default value is IDI (for initial domain identifier).

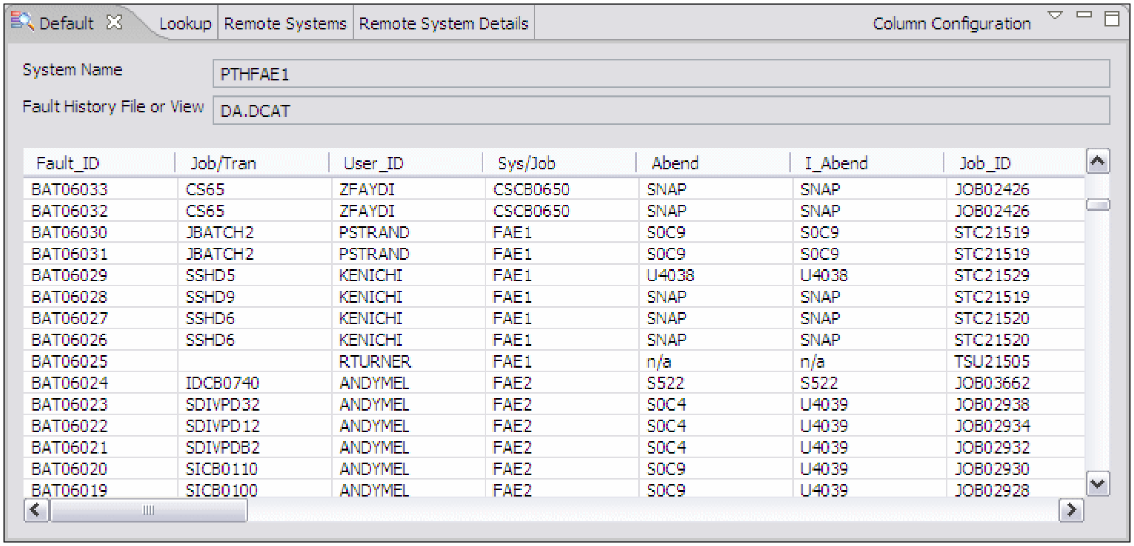
Table 6-7 provides a summary of all available actions in the FA Artefacts view.

Table 6-7 FA Artefacts view actions

Action name	Description
Add new history file	Enables you to add a history file from a particular host to the view.
Retrieve last accessed history file	Enables you to retrieve the set of history files that you last used on the host, based on the information stored in your ISPF profile.
Retrieve view information	Enables you to retrieve information about views, which are stored in a data set.
Delete from view	Deletes the selected item from the view.
Populate/refresh history file	Refreshes the cached information about the selected history file.
Configure history file monitor	Allows you to configure a history file monitor to monitor the changes that are made to a history file on the host. Whenever changes are made to the selected history file, the user is notified.
Set encoding	Enables you to specify the encoding for the selected system, that is, the language option that was used when the fault entry was created.

6.6.2 Default view

The Default view (Figure 6-10) shows a detailed summary of fault entries contained in the selected history file or view in the FA Artefacts view. This view gives you a quick summary of what is happening on your system (history file-centric view of the system). You can configure the column headings, depending on your preference and area of interest.



Fault_ID	Job/Tran	User_ID	Sys/Job	Abend	I_Abend	Job_ID
BAT06033	CS65	ZFAYDI	CSCB0650	SNAP	SNAP	JOB02426
BAT06032	CS65	ZFAYDI	CSCB0650	SNAP	SNAP	JOB02426
BAT06030	JBATCH2	PSTRAND	FAE1	S0C9	S0C9	STC21519
BAT06031	JBATCH2	PSTRAND	FAE1	S0C9	S0C9	STC21519
BAT06029	SSHD5	KENICHI	FAE1	U4038	U4038	STC21529
BAT06028	SSHD9	KENICHI	FAE1	SNAP	SNAP	STC21519
BAT06027	SSHD6	KENICHI	FAE1	SNAP	SNAP	STC21520
BAT06026	SSHD6	KENICHI	FAE1	SNAP	SNAP	STC21520
BAT06025		RTURNER	FAE1	n/a	n/a	TSU21505
BAT06024	IDCB0740	ANDYMEL	FAE2	S522	S522	JOB03662
BAT06023	SDIVPD32	ANDYMEL	FAE2	S0C4	U4039	JOB02938
BAT06022	SDIVPD12	ANDYMEL	FAE2	S0C4	U4039	JOB02934
BAT06021	SDIVPDB2	ANDYMEL	FAE2	S0C4	U4039	JOB02932
BAT06020	SICB0110	ANDYMEL	FAE2	S0C9	U4039	JOB02930
BAT06019	SICB0100	ANDYMEL	FAE2	S0C9	U4039	JOB02928

Figure 6-10 Default view of fault entry summaries

From this view, you can invoke the Report browser (see 6.6.4, “Fault Analyzer Report browser” on page 325). The Report browser enables you to see the detailed analysis of the selected fault entry or to invoke the Dump browser (see 6.6.5, “Fault Analyzer Dump browser” on page 327) to browse a hex dump display of mini-dump pages stored in the selected fault entry.

Table 6-8 shows a summary of all the actions that are available from this view.

Table 6-8 Fault Analyzer Report browser action

Action name	Description
Browse report	Retrieves the report associated with the selected fault entry and shows the report. The report is cached locally after it is retrieved.
Browse mini-dump	Retrieves the mini-dump pages associated with the selected fault entry and shows the pages. When the pages associated with a fault entry are retrieved, they are cached locally.
Delete cached data	Deletes the cached information for the selected fault entry.
Delete from view	Deletes the selected item from the view.
Set encoding	Allows you to select an encoding for the system where the selected history file or view is located.

6.6.3 Working with older fault entries

For a Fault Analyzer Integration client to show fault entries in a history file correctly, you must create the fault entries with the correct version of the Fault Analyzer feature, with the appropriate set of options specified.

Whenever the Fault Analyzer plug-in detects a fault entry that does not contain information required by the Fault Analyzer client, it offers an option to refresh the fault entry (Figure 6-11). That is, you can perform the analysis again to re-create the contents so that it can be viewed using the client. If you select to refresh the fault entry, the Rational Developer for System z cached information associated with the fault entry is deleted, the fault entry is recreated, and the contents of the fault entry are downloaded to the client for display.

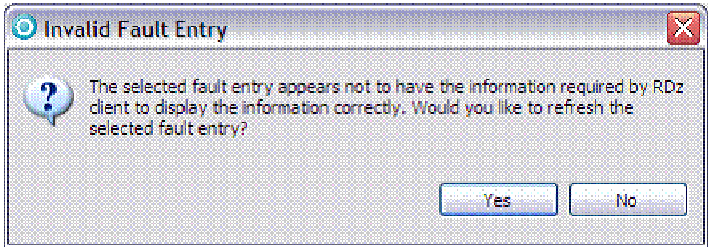


Figure 6-11 Invalid Fault Entry pop-up

6.6.4 Fault Analyzer Report browser

The Fault Analyzer Report browser (Figure 6-12) is a multi-tabbed browser that shows the report that is associated with the selected fault entry. You typically invoke the browser by running the Browse Report action in the Default view.

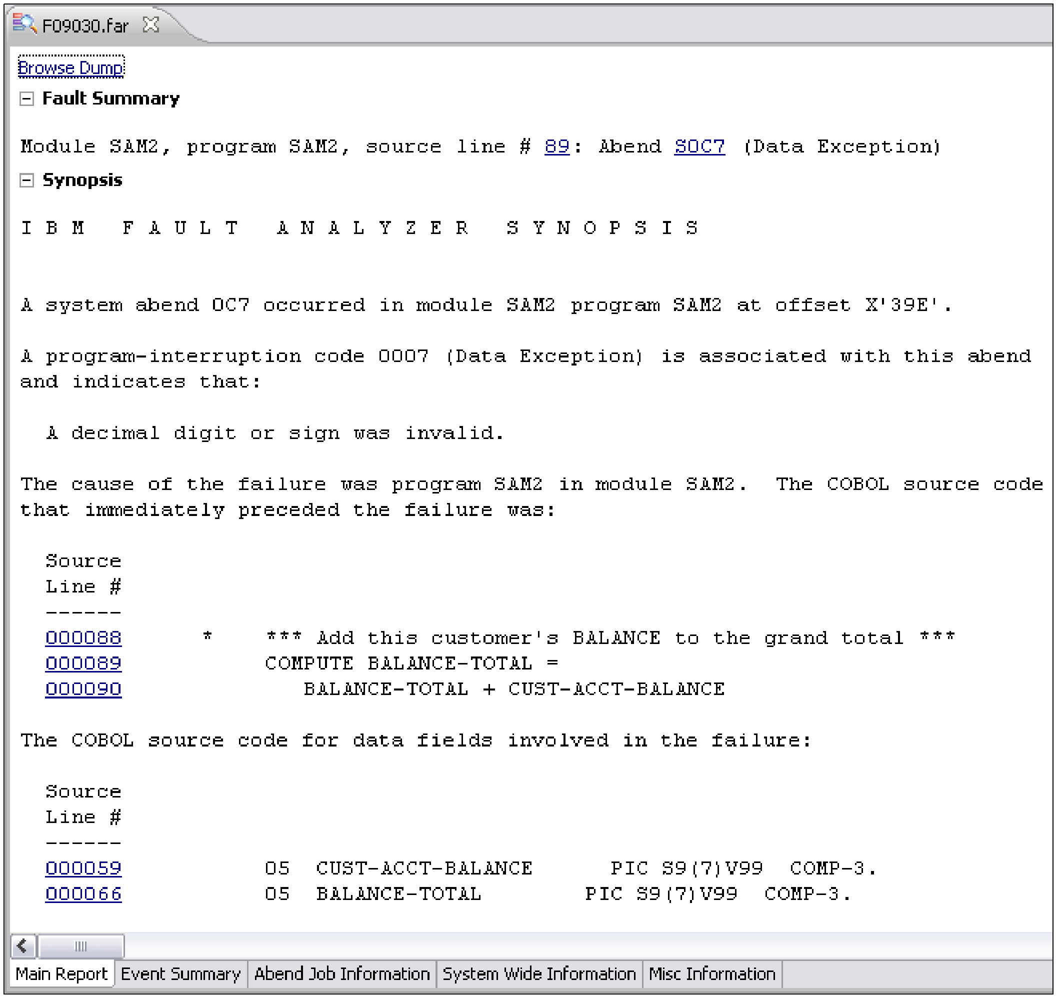


Figure 6-12 Fault Analyzer synopsis of an ABEND

You can look at separate parts of the report by using the tabs that show along the bottom of the window. The tabs separate the report into these five categories and related subcategories:

- Main Report:
  - Fault summary section: A brief description of the fault, which includes the hyperlinked source line number of the ABENDing program (if possible) and the ABEND code. If you click the source line number, it shows the source code of the ABENDing program where the failure occurred (see 6.6.6, “Source code display” on page 329 for further information). If you click the ABEND code, it shows the description.
  - Synopsis section: The synopsis section provides a brief description of the fault and its analysis.
- Event Summary:
  - The Event Summary is a list of all events, in chronological order.
  - The Event details subsection provides detailed information about each event. This subsection includes additional information associated with the event, such as the message description and the contents of the program’s working storage. This subsection also includes the source code information or failing machine instruction.
- ABEND Job Information:
  - This section provides information about the ABENDing job that is associated with the real-time invocation of the Fault Analyzer feature, such as:
    - ABEND date
    - Time
    - Job ID
    - Job name
    - Job step name
    - Execution environment
    - Language environment runtime options
- System-Wide Information:
  - This section contains, for example, console messages that are not identified as belonging to any specific event or CICS system-related information, such as trace data and 3270 screen buffer contents. This section also includes any information about open files that cannot be associated with any specific event.
  - If there is no system-wide information available, this section does not appear in the report.

- Miscellaneous Information:
  - Fault Analyzer options: Lists of the Fault Analyzer options that were in effect at the time of the analysis.
  - Prolog section: Shows information about the version, release, and modification level of the Fault Analyzer, as well as the latest authorized program analysis report (APAR) or program temporary fix (PTF) installed.
  - Epilog section: Provides information about the invocation exits used and the approximate amount of above-the-line storage allocated during the analysis, followed by the fault ID assigned. It also includes the time and date when the report was created.

See the *Fault Analyzer for z/OS V8R1 User's Guide and Reference*, SC19-1253-05, for a comprehensive description.

6.6.5 Fault Analyzer Dump browser

Fault Analyzer Dump browser (Figure 6-13) is a multi-tabbed browser that enables you to browse the mini-dump pages that are stored for the selected fault entry in a history file. This browser is typically opened by selecting the **Open Mini-Dump** action against a fault entry in the summary view. *Mini-dump pages* are the virtual storage pages that are accessed during analysis and stored as part of a fault entry.

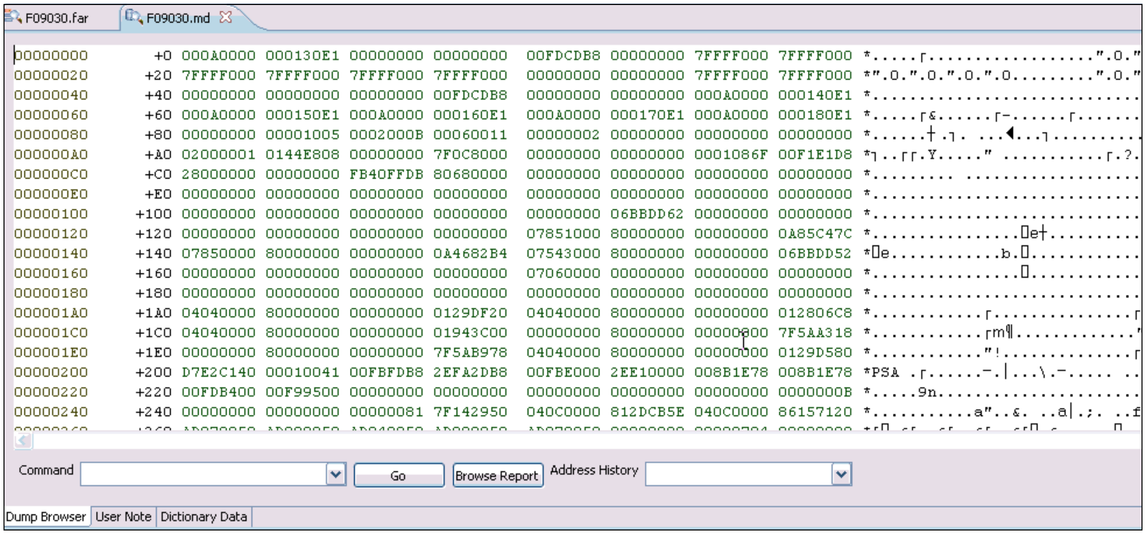


Figure 6-13 Mini-dump

Mini-dump pages are annotated with the key information that the Fault Analyzer found during the analysis (dictionary data). This dictionary data includes information, such as various control blocks, modules, Multiple Virtual Storage (MVS) areas, and register values. You can see a summary of the dictionary data that is associated with the selected fault entry in the Outline view (Figure 6-14) or in the Dictionary Data tab within the Dump browser.

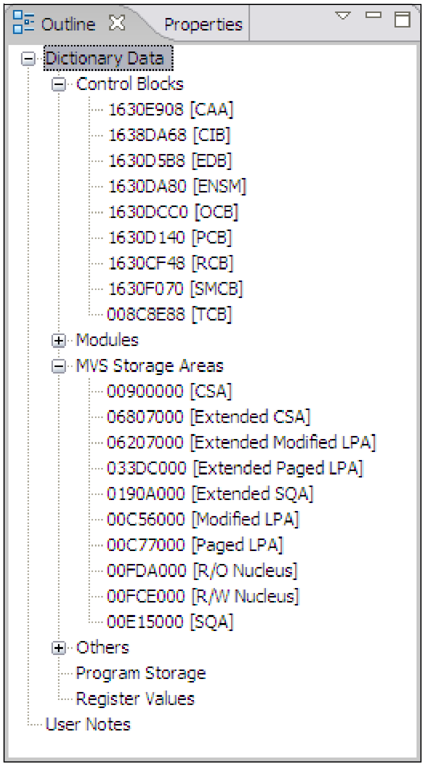


Figure 6-14 Mini-dump Outline view

6.6.6 Source code display

Fault Analyzer supports several side file formats to display the associated source code information of an ABENDing program.

The Fault Analyzer Integration option supports the source line display of the ABENDing program if the correct side file was available during the analysis. If the correct side file was available during the analysis of your ABENDing program, the source line number of the program is hyperlinked in the report. When you click the source line number, it retrieves the side file from the host and generates the source line for display. The source file opens in your favorite editor in read-only mode.

You can specify the data set names (DSN) containing your side files during the fault entry refresh process (Figure 6-15).

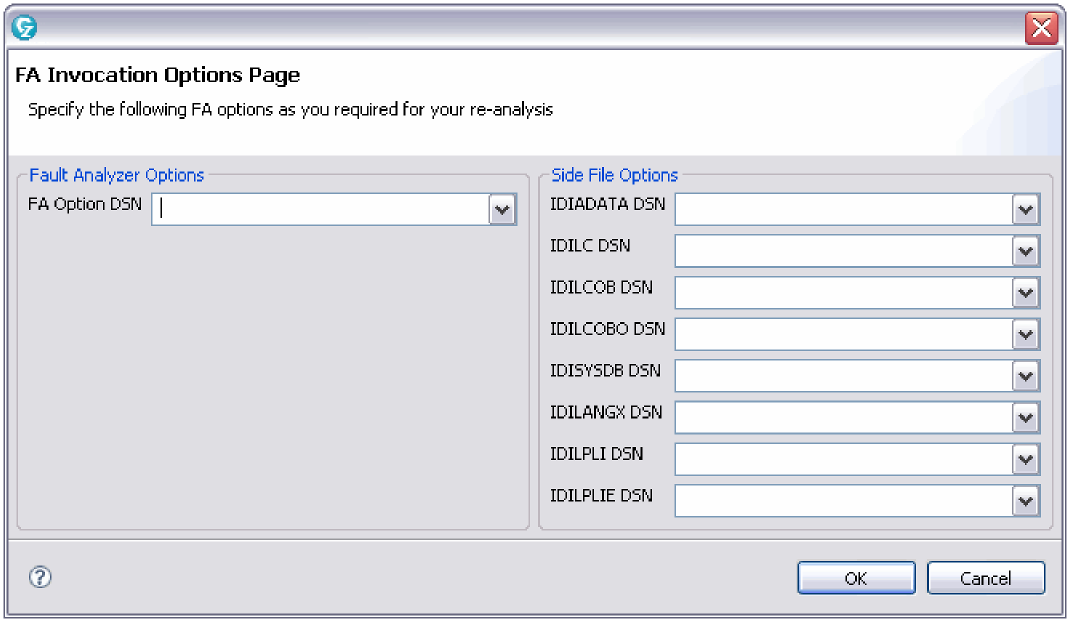


Figure 6-15 FA Invocation Options Page

Table 6-9 describes the Fault Analyzer actions.

Table 6-9 FA invocation option descriptions

Action name	Description
FA Option DSN	A fully qualified data set name, which contains the Fault Analyzer options (usually SYS1.PARMLIB(IDICNF00)). Refer to *Fault Analyzer for z/OS V8R1 User's Guide and Reference*, SC19-1253-05, for further details.
IDIADATA DSN	The name of one or more sequential or partitioned data set extended (PDSE) data sets holding Assembler SYSADATA files.
IDILC DSN	The name of one or more sequential or PDSE data sets holding C compiler listings.
IDILCOB DSN	The name of one or more sequential or PDSE data sets holding COBOL compiler listings (other than OS/VS COBOL).
IDILCOBO DSN	The name of one or more sequential or PDSE data sets holding OS/VS COBOL compiler listings.
IDISYSDB DSN	The name of one or more sequential or PDSE data sets containing COBOL or Enterprise PL/I SYSDEBUG side files. These side files are created when compiling a COBOL program with the TEST(,,SEPARATE) options.
IDILANGX DSN	The name of one or more sequential or PDSE data sets holding side files.
IDILPLI DSN	The name of one or more sequential or PDSE data sets holding PL/I compiler listings (other than Enterprise PL/I).
IDILPLIE DSN	The name of one or more sequential or PDSE data sets holding Enterprise PL/I compiler listings.

6.6.7 Lookup view

In the Lookup view, you can browse to the description about ABEND codes, messages, and other miscellaneous information. Figure 6-16 shows the description of IMS user ABEND U0002. You can browse the information in one of three ways:

- Find the description for a particular ABEND code, message, or miscellaneous information by navigating through the tree structure.
- Type a pattern name in the Search box. All matching ABEND codes, messages, and miscellaneous information show in the Results tab in the view. From the list of matching results, you can browse to find the description.
- An ABEND code associated with a particular fault entry is hyperlinked in the report view. Click the hyperlinked ABEND code to display the description.

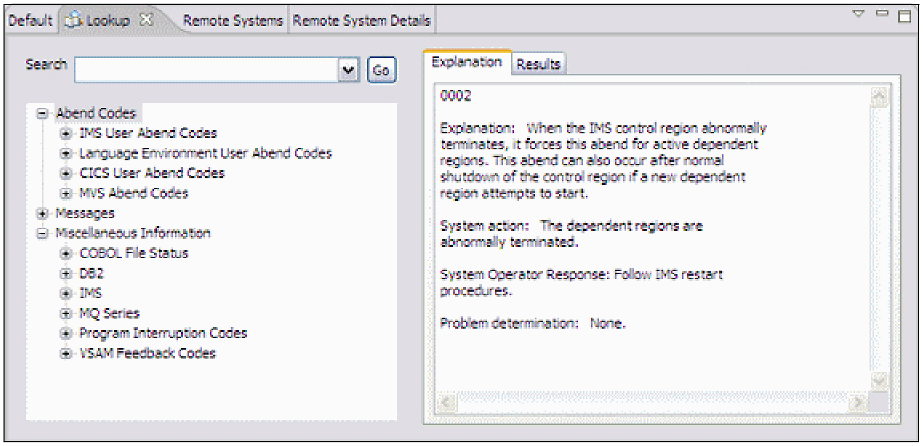


Figure 6-16 Lookup view

You can switch between English or Japanese descriptions for ABEND codes, messages, and miscellaneous information:

1. Select the **Language Setting** action from the drop-down menu within the view.
2. When the Lookup Language Setting dialog appears, choose your preferred language.

6.7 IBM File Manager for z/OS

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

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

6.8 Using File Manager

IBM File Manager for z/OS is a member of the IBM PD Tools suite, which provides comprehensive tools for working with IBM OS/390® data sets, DB2 data, IMS data, CICS data, and MQ data. These tools include the familiar browse, edit, copy, and print utilities that are found in ISPF, which are enhanced to meet the requirements of application developers.

The IBM Rational Developer for System z File Manager feature integrates with IBM File Manager for z/OS to provide application developers with the ability to browse and edit sequential data sets, partitioned data set members, and Virtual Storage Access Method (VSAM) data sets of the key-sequenced data set (KSDS) and entry-sequenced data set (ESDS) types in a workstation Rational Developer for System z environment. The IBM Rational Developer for System z File Manager feature can show records that are formatted according to the field layouts that are typically provided by COBOL copybooks or PL/I include files.

File Manager integration facilitates access to structured data files from the Rational Developer workbench, including:

- Support for the template-driven display of VSAM, PDS member, and sequential file data. Rational Developer now offers full support for creating, editing, and updating existing or new File Manager templates, providing equivalent functionality to the green screen interface.
- Browsing and updating data sets from the workbench. Rational Developer provides a window to the data set with which you are working, similar to the green screen interface, but enhanced with additional search and data manipulation actions:
  - Support for large data set display and edit. File Manager Integration does not download the entire data set to the workstation, only the part of the data set with which you work. Rational Developer can still facilitate actions against the entire data set using the File Manager engine on the host.
  - Support for searching large data sets through host-based searching and display.
  - New VSAM utility operations, such as copy and allocate, that are similar to the utility operations that are available from the Remote Systems Explorer browser

In this section, we demonstrate the features present in Rational Developer File Manager to edit a data set. We explain the steps that are required to configure Rational Developer File Manager so that it can communicate with IBM File Manager. Then, we provide an example of using the Rational Developer File Manager Formatted Data Editor to edit a KSDS.

6.8.1 Editing data set records using Rational Developer File Manager

In this section, we demonstrate the flow of browsing and editing records that are stored in a data set using Rational Developer File Manager. In particular, we demonstrate how to specify the record format of the data that is stored in a KSDS data set using a COBOL copybook. The steps that we describe in this section are similar to the steps for other types of data sets, such as VSAM files. The steps are similar to the steps you use when specifying the record format of the data using a PL/I include or IBM File Manager for z/OS template instead of a COBOL copybook.

Specifying z/OS file mapping for the sample data set

When you choose to browse a data set, in the Remote Systems view under MVS files, you can see a list of data sets that are listed with various icons to indicate the separate types of data sets and their associated editors. The z/OS File System Mapping view specifies the association of a data set name and its preferred editor (Figure 6-17).

By default, File Manager Integration gives mainframe data sets, which match the pattern `**DATA`, the `tdat` extension when being displayed in the Remote Systems view. If you browse the file associations for `tdat` files (select **Window** → **Preferences** → **General** → **Editors** → **File Association**), they have Formatted Data Editor, by default. The data within these files shows with the Rational Developer File Manager Formatted Data Editor, by default. In addition to editing `tdat` files, you can use the Formatted Data Editor to edit VSAM clusters, whether or not they have the `tdat` extension, because this editor is the only Rational Developer editor that is currently capable of editing VSAM cluster data.

If the default mappings do not match the naming conventions of your data sets, you can add custom mappings, as well. It is unnecessary to add custom mappings in this case, because the data set that we have created is a VSAM cluster and it matches the `**DATA` pattern.

Both of these conditions cause the data set to open in the Formatted Data Editor, but we add a mapping as an example.

In the z/OS File System Mapping view, right-click and select **Add Data Set Mapping**. Enter `**KSDS` (or a fully qualified data set name, if preferred), as the mapping criterion and `tdat` in the Other text box under Workstation File Extension. If data sets display that still do not have the correct workstation file extension, you can force the file extension by editing the Extension field in the Mapping view of the data set's properties dialog.

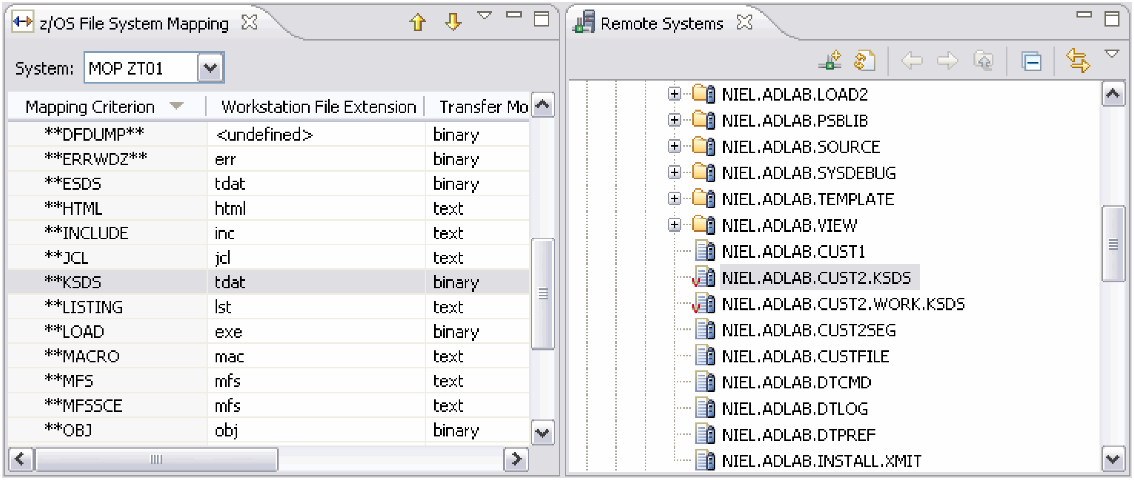


Figure 6-17 Data set mapping view

Specifying the data set's record layout

The Formatted Data Editor must know the format of the records that are stored in a data set to display the contents in a meaningful way for application developers. For this example, we use a COBOL copybook to describe the data. Right-click the VSAM data set. From the context menu, select **Properties**, which brings up the Properties dialog for the selected data set, as shown in Figure 6-18. Select **Template Association**. In this view, you can either specify the full name of the template to use for the selected data set or you can browse through the available data sets in your Remote Systems filter for the current target system.

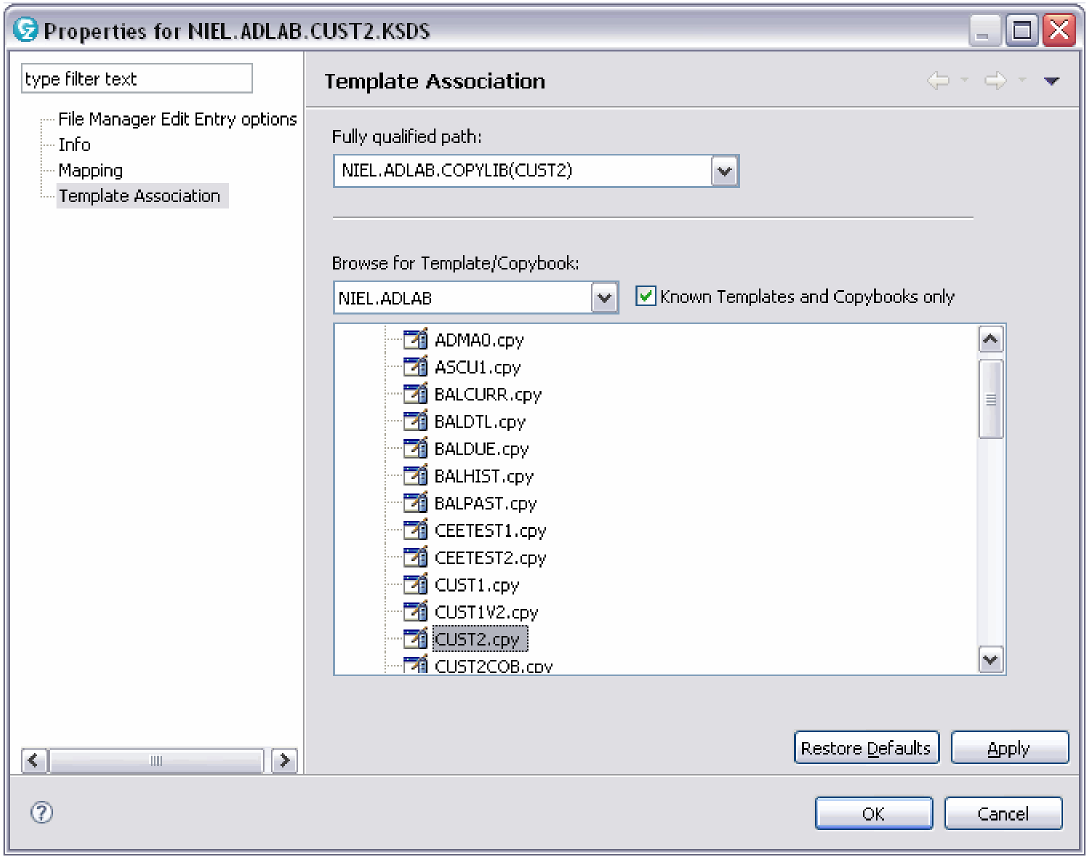


Figure 6-18 Data set layout selection

In addition to using COBOL copybooks to specify the record layout, you can use PL/I include files and IBM File Manager templates. File Manager templates are derived from COBOL copybooks, and PL/I includes them by compiling them under ISPF. They can contain additional information, such as identification and selection criteria.

While browsing for templates in the Template Association window (Figure 6-20 on page 337), you can select **Known Templates and Copybooks only** to filter out all data sets that do not have the workstation file extensions currently for COBOL copybooks, PL/I includes, or IBM File Manager templates. The Rational Developer File Manager user interface and documentation uses *template* as a generic term to refer to all three of these data set types. Refer to the *File Manager for z/OS and OS/390 V2R1 User's Guide and Reference,* SC27-1315-01, for further information about templates.

Using a copybook to describe the record format

In this step, we view the COBOL copybook that describes the format of the records that are stored in the sample data set. Figure 6-19 shows the sample COBOL copybook in the Rational Developer System z LPEX Editor.

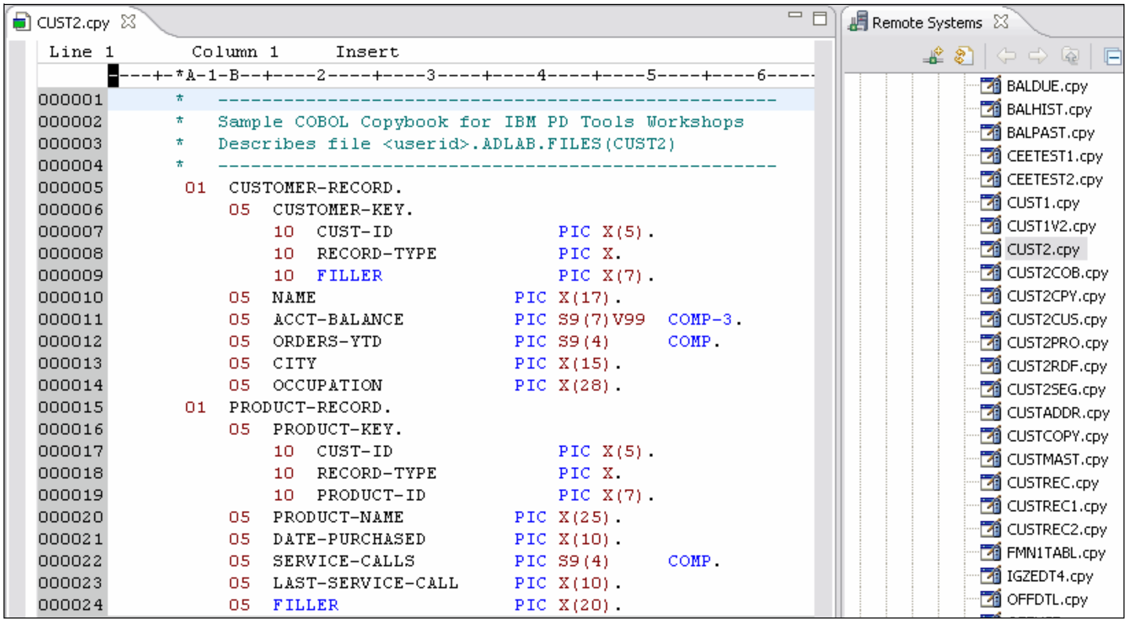


Figure 6-19 LPEX Editor

6.8.2 Editing the sample data set in the Formatted Data Editor

In this section, we describe editing a data set using the Formatted Data Editor.

Opening a data set in the Formatted Data Editor

Right-click the data set to open. Select **Open With** → **Formatted Data Editor**. A progress window opens while an IBM File Manager for z/OS edit session starts for the requested data set. After the IBM File Manager for z/OS edit session begins and the Formatted Data Editor retrieves the initial data for the edit session, the Formatted Data Editor displays (Figure 6-20).

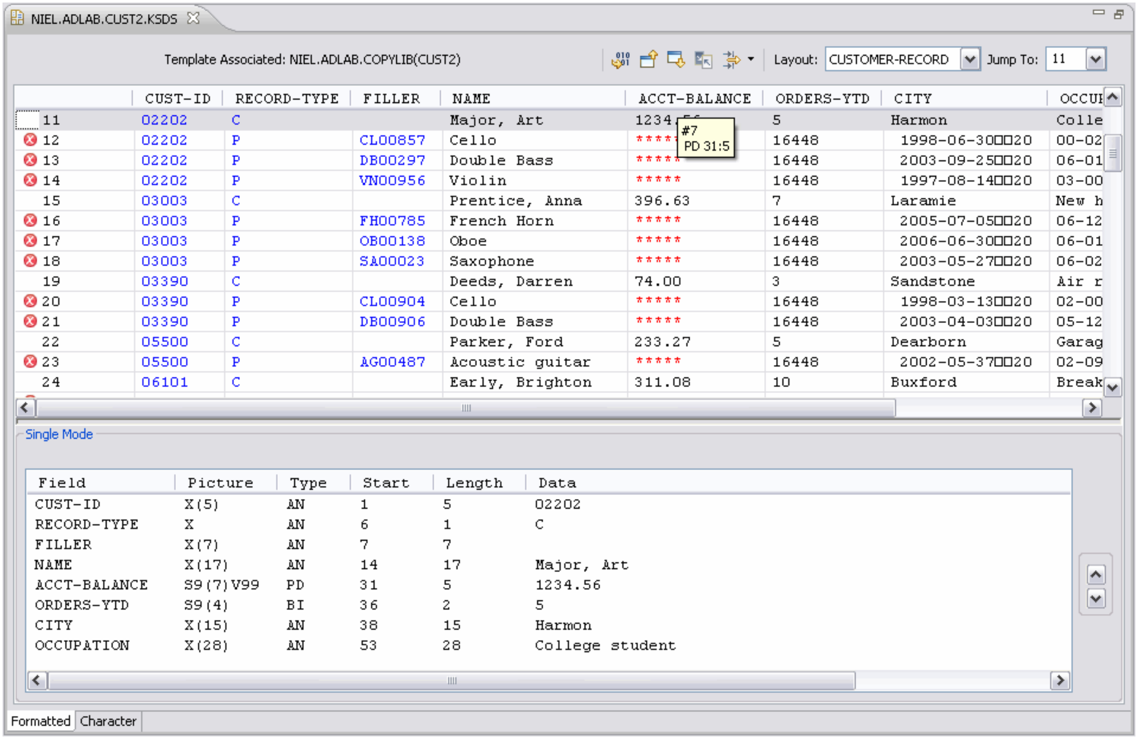


Figure 6-20 Data Editor

Using table and single modes

Notice the following characteristics in the default Formatted Data Editor interface (Figure 6-20):

- After `Template Associated:`, you see the name of the template that provides the record layout for the session.
- A tabular view of the data shows underneath the name of the template that has been associated. This view is the *table mode*:
  - Each row in the table represents one record in the data set.
  - The fields within each record are formatted correctly in columns with the appropriate headings for each column based on the information that is specified in the COBOL copybook.
  - Records and fields that do not match the current record layout are marked as incorrect.
  - As defined by the VSAM KSDS, the fields containing the KSDS key fields display in blue instead of black.
- One record's data shows at a time in the lower part of the Formatted Data Editor. This view is the single mode:
  - All modes are synchronized. Changes that are made in one mode are immediately reflected in all the other modes.
  - You can remove the single mode from view by clicking the control button. You can change the single mode's size with respect to the table mode by dragging the divider between them up or down.
  - The up and down buttons on the right side of the single mode move the selected record up or down. The change in selection also shows in the table mode.

Using character mode

You can see two tabs in the lower-left corner of the editor. The Table/Single tab is the default. By clicking the **Character** tab, you can enable the *character mode*. When in character mode (Figure 6-21), the records appear as though they only contain alphanumeric data.

The record layout that is described in the template is not applied to the records. If the data in the records is not alphanumeric and cannot be displayed as alphanumeric, a box character appears to represent the character that cannot be displayed (Figure 6-21). If a data set is opened without a template associated with it, the table mode shows data in the same way that the character mode shows data for all data sets. Because the character mode is not as useful for simple editing operations as the table and single modes, we return to the Table/Single mode tab for the rest of the example.

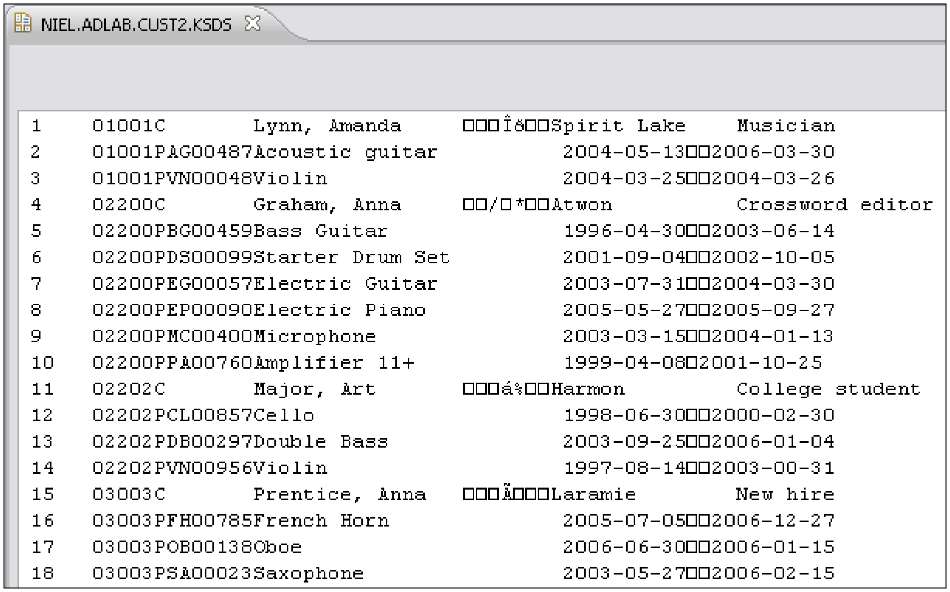
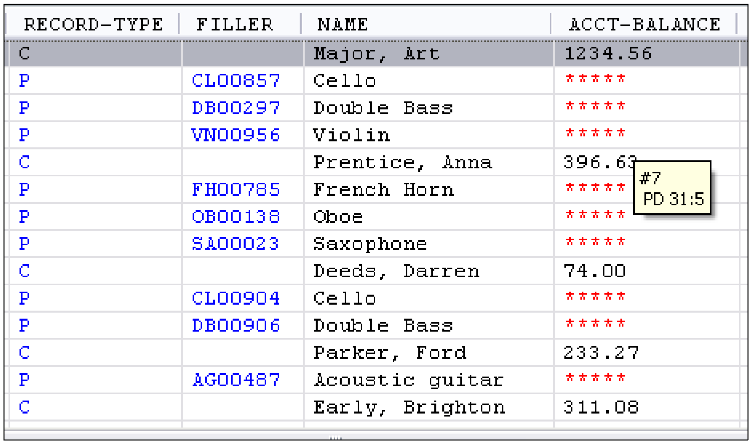


Figure 6-21 Character mode

Field metadata

In the table and single modes, the record layout field metadata shows when the mouse hovers over the field data. In Figure 6-22, we see that field number 7 (#7), ACCT-BALANCE, from the associated copybook is a packed decimal field (PD), starts at position 31 in the data set, and is 5 bytes long.



RECORD-TYPE	FILLER	NAME	ACCT-BALANCE
C		Major, Art	1234.56
P	CL00857	Cello	*****
P	DB00297	Double Bass	*****
P	VN00956	Violin	*****
C		Prentice, Anna	396.63
P	FH00785	French Horn	*****
P	OB00138	Oboe	*****
P	SA00023	Saxophone	*****
C		Deeds, Darren	74.00
P	CL00904	Cello	*****
P	DB00906	Double Bass	*****
C		Parker, Ford	233.27
P	AG00487	Acoustic guitar	*****
C		Early, Brighton	311.08

Figure 6-22 Field metadata

Editing a field

To edit a field, simply select it with the mouse and modify the contents. In the current example, changes to field #7 must be numeric. If invalid data is entered for a given field type, the error shows near the field, as shown in Figure 6-23, and the record is marked as an error. If there is an error in the entered data, you cannot save the change to the field.

	CUST-ID	RECORD-TYPE	FILLER	NAME	ACCT-BALANCE	ORDERS-YTD
11	02202	C		Major, Art	A1234.56	5
12	02202	P	CL00857	Cello	*****	
13	02202	P	DB00297	Double Bass	*****	16448
14	02202	P	VN00956	Violin	*****	16448
15	03003	C		Prentice, Anna	396.63	7
16	03003	P	FH00785	French Horn	*****	16448
17	03003	P	OB00138	Oboe	*****	16448
18	03003	P	SA00023	Saxophone	*****	16448
19	03390	C		Deeds, Darren	74.00	3
20	03390	P	CL00904	Cello	*****	16448
21	03390	P	DB00906	Double Bass	*****	16448
22	05500	C		Parker, Ford	233.27	5
23	05500	P	AG00487	Acoustic guitar	*****	16448
24	06101	C		Early, Brighton	311.08	10

Invalid character in field

Figure 6-23 Data error

If the new contents of the field are valid, the change is allowed and maintained. For this example, we now change record 11's CUST-ID field from `02202` to `02203` (Figure 6-24). Note that the single mode has been updated to show the new values for the record as expected and that the name of the data set on the editor tab in the upper left corner has an asterisk beside it. The asterisk denotes a "*dirty*" editor. When the editor is dirty, changes that have been made to the data are pending, but the changes have not been saved back to the actual data set. Until a save operation is requested (select **File** → **Save**), the data set and the data in the current edit session are out of sync.

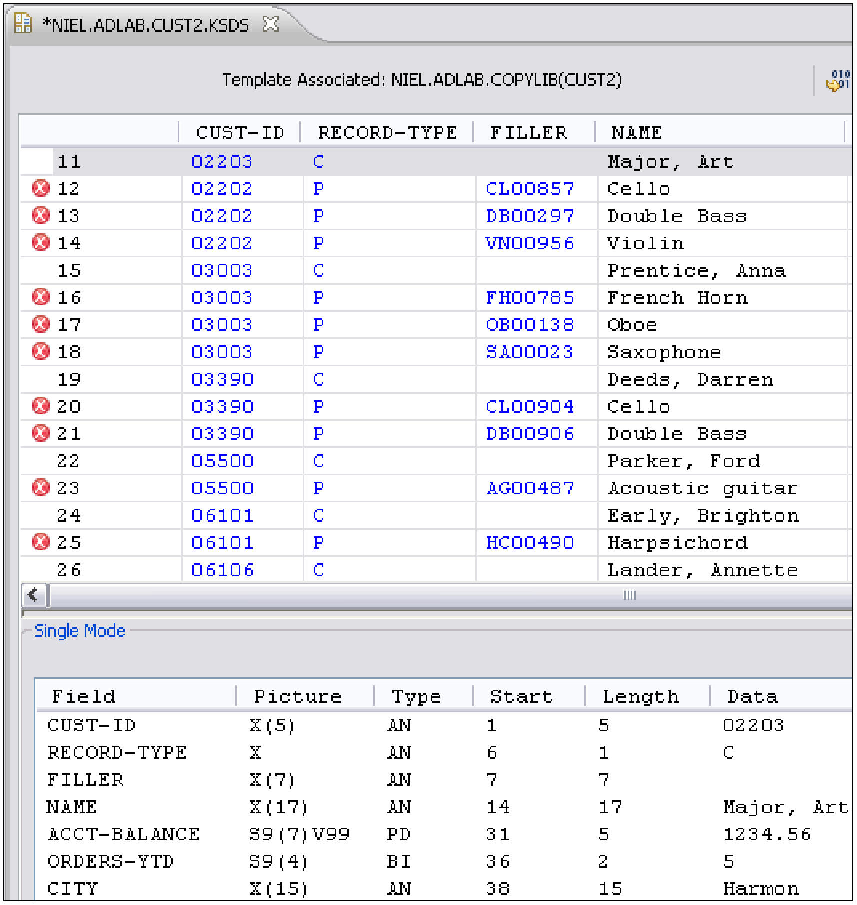


Figure 6-24 Unsaved changes in the Formatted Data Editor

Editing hex

You can display and edit the underlying hexadecimal data by clicking **Hex**. The hexadecimal representation of the data shows beneath each record, so that the two-byte nibbles are directly underneath the respective byte in the major record display (Figure 6-25). This presentation is the same as the hexadecimal displayed by default in IBM File Manager.

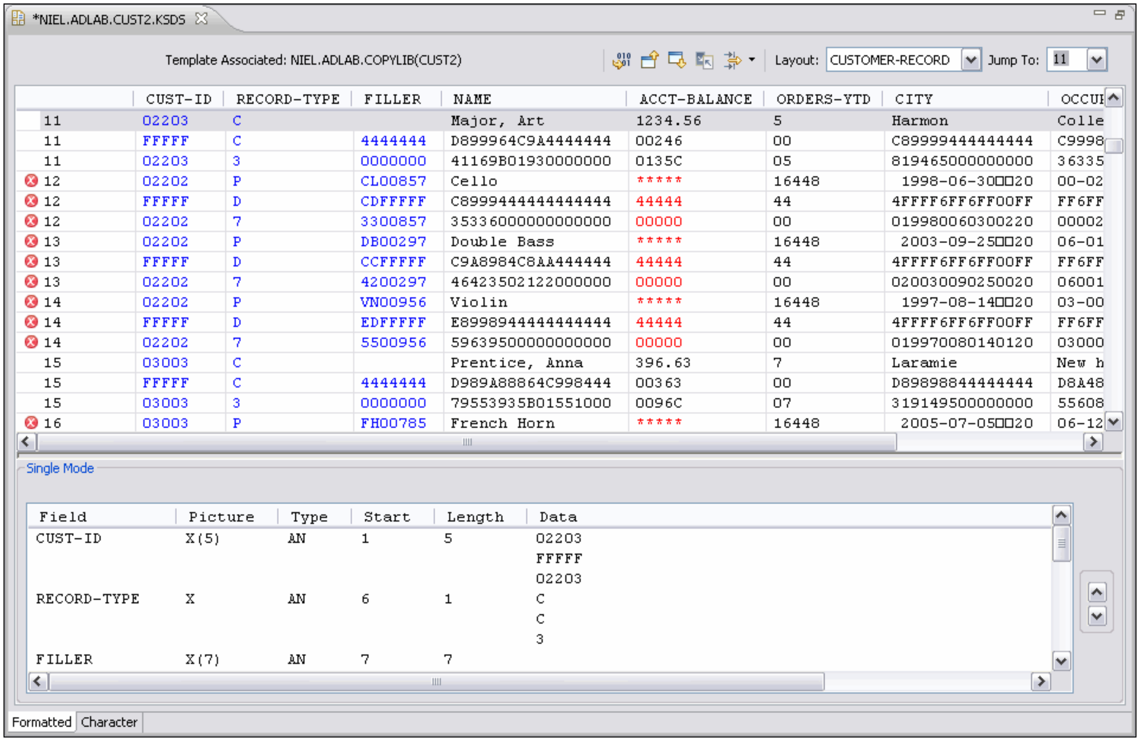


Figure 6-25 Hex Data Editor

You can edit the hexadecimal representation. You can even force invalid values for field types (for example, an alphabetic character in our packed decimal field) into a field by editing the hexadecimal representation. If a field contains an invalid value for its specified type, its display representation becomes a series of asterisks.

Using the Template Editor

The *Template Editor* is an area in which you can work directly with templates, independently of specific data sets. You can use this editor to create and modify dynamic or regular COBOL and PL/I copybook templates.

The Template Editor offers various options for creating new templates and editing existing templates for use within the Formatted Data Editor. Templates improve the functionality of the Formatted Data Editor by providing a particular interpretation of a data set. The Template Editor allows you to specify the particular interpretation of a data set by providing the necessary tools to create and modify templates easily. This functionality gives you additional control over the way that you interact with your data.

The File Manager Integration Template Editor provides a similar level of editing functionality as is available in IBM File Manager. For more information about the IBM File Manager ISPF Template Editor, see the *File Manager for z/OS and OS/390 V2R1 User's Guide and Reference,* SC27-1315-01.

To create a new template, right-click in the Remote Systems Explorer and Select **File Manager** → **Create Template**, as shown in Figure 6-26.

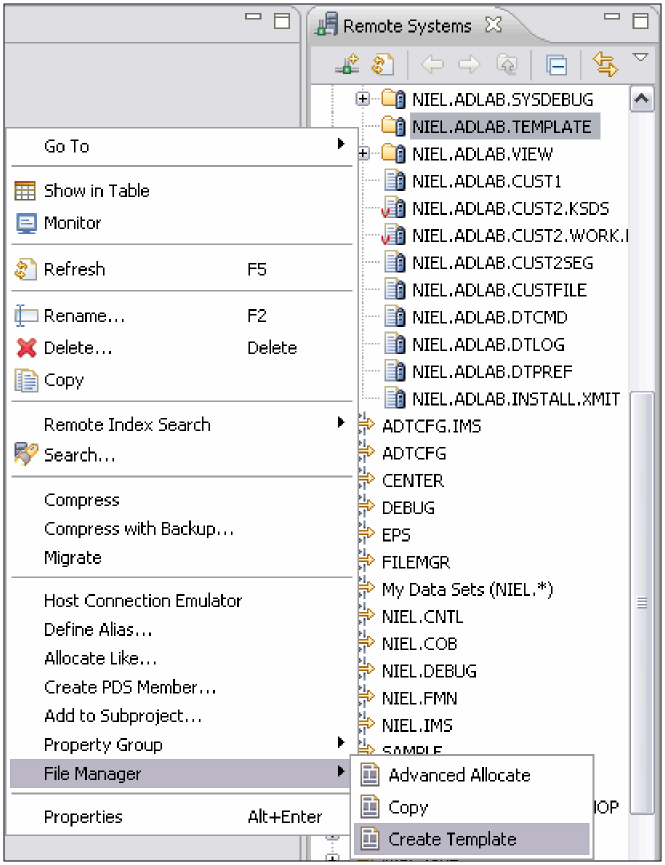


Figure 6-26 Create a new file manager template

The Create Template wizard prompts you for a new Template name and, if you are creating the template from a copybook, the location of a copybook member. Then, you can use the Template Editor (Figure 6-27) for defining specific layout characteristics, such as the offset of a record layout, as well as various methods for defining record criteria.

Use the options that are located at the top of the Template Editor to identify if the currently selected template will be used to access segmented data. Also, if your template contains more than one record layout, use the drop-down list box to select which record layout to edit (Figure 6-27).

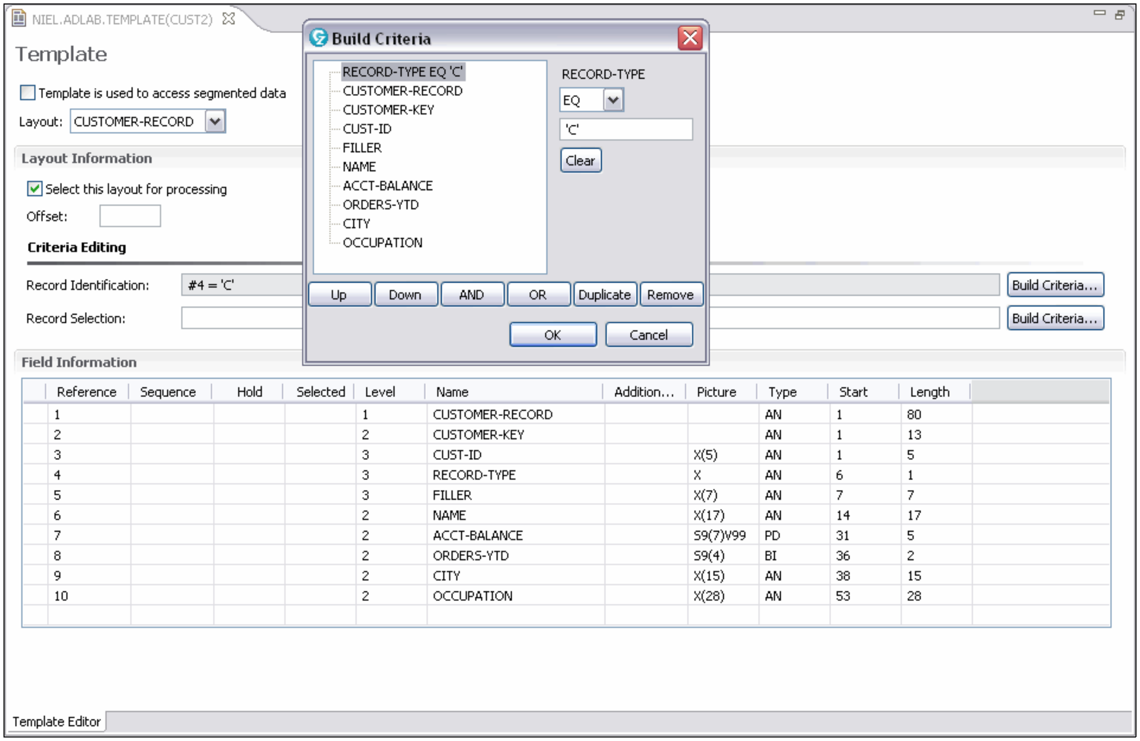


Figure 6-27 Specifying record identification criteria

You can use the Template Editor to select specific records that you want to work with from a data set. Depending on the type of template being edited, you can define three types of record criteria when using the Template Editor:

- Related record identification criteria
- Record identification criteria
- Record selection criteria

You can define the criteria for these three types by using two separate methods: the freeform method or the by-field Build Criteria editor.

Record identification criteria identifies the separate record types within a data set. If your copybook or template contains multiple record types, each type is initially distinguished by the record length. If each type has a unique record length, you can select and display the various types within the Formatted Data Editor.

However, if two or more record types have the same length, you can use record identification criteria to be able to view the various types. You define the record identification criteria within templates using the Template Editor. After you specify the record identification criteria within a template, you can apply it against a data set to identify separate types of records that are present in the data set.

When specifying this type of criteria, first examine your data sets and then determine what field values uniquely identify a record with a particular type. You can use this method to match records within a data set to a particular record layout. By default, record length is used as an identification criteria if nothing else has been specified. If the default is used and the length of a record matches more than one record type within a template, the first matching record type is used.

You can create and modify record identification criteria by using the tools that are provided in the Layout information portion of the Template Editor. After each record of the data set can be identified correctly, you can display all the record types using the Layout drop-down list, and you can select the record type that you want (Figure 6-28).

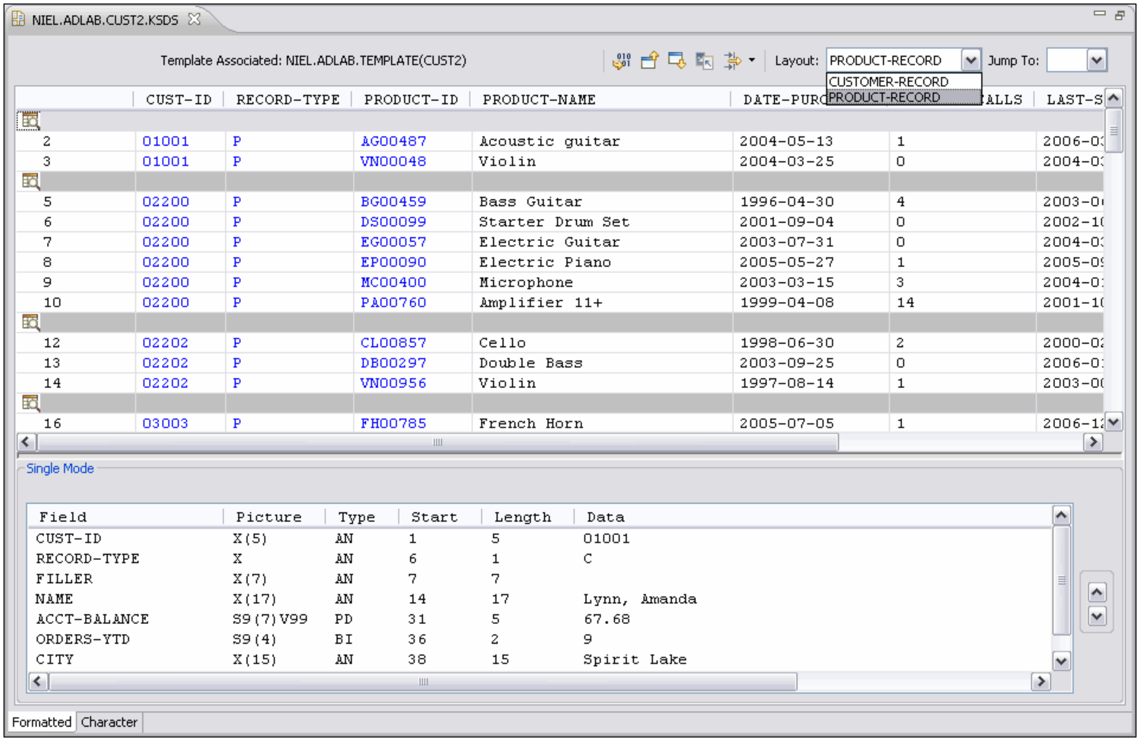


Figure 6-28 Formatted Data Editor using selectable record types

Supporting extremely large data sets

Certain data sets might be so large that they do not fit in the workstation's available system memory. To allow you to edit these large data sets, the Formatted Data Editor pulls a subset of the records into the target data set at a time.

The records that are currently in the editor are known as the *editor's window* of the data set. The default window size is 200 records. There are two buttons at the top of the Formatted Data Editor that you can use to move the editor's window of the data set. Moving the window up or down is also referred to as "*stepping*" through the data set (Figure 6-29). By default, the window step size is 50 records.

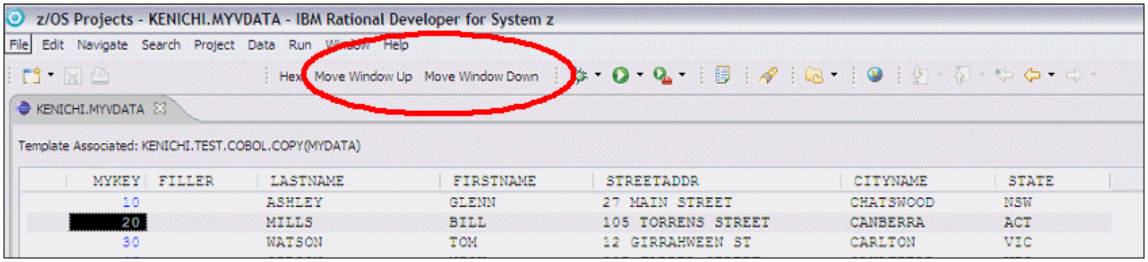


Figure 6-29 Moving the window up or down in the Formatted Data Editor

The current position in the data set shows at the bottom of the editor. Figure 6-30 shows that records 21 - 40 out of 774 records are currently being displayed.

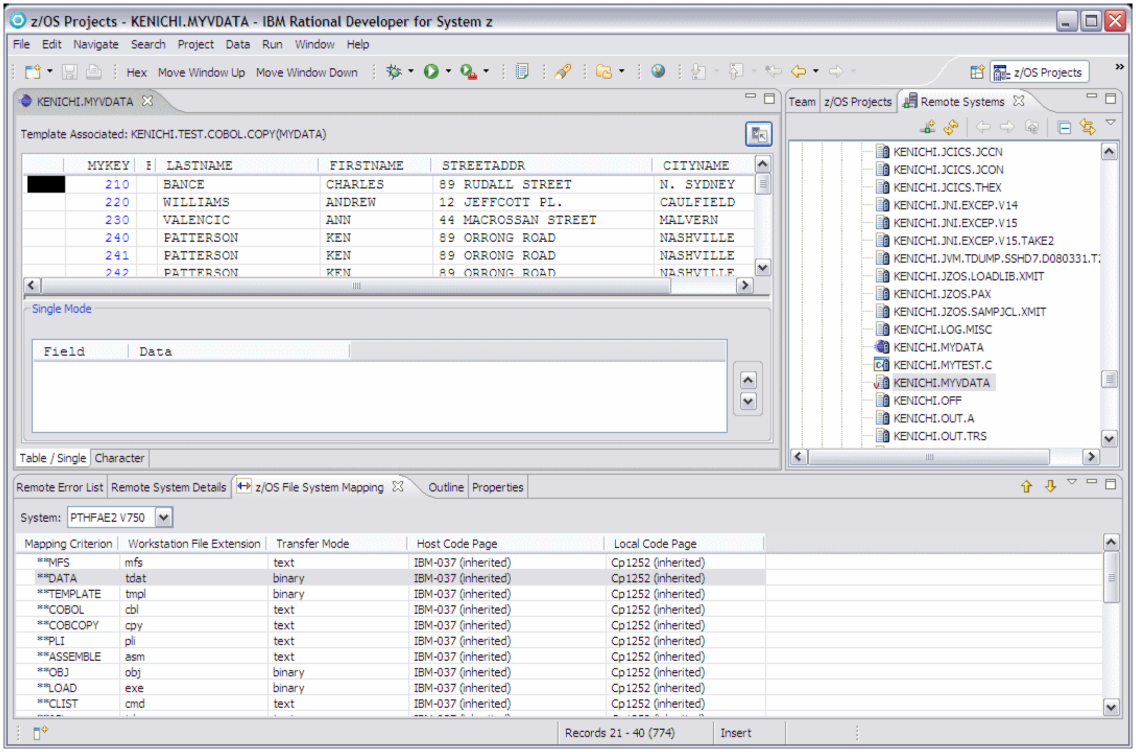


Figure 6-30 Displaying data

6.8.3 File Manager components

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

Base component

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

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

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

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

DB2 component

If you are a DB2 database administrator (DBA), an application programmer, or you just want to retrieve information that is stored in a DB2 database, the File Manager DB2 component provides assistance for you.

The File Manager DB2 component provides database management capabilities, such as creating and dropping objects, copying data within DB2, handling privileges, importing, and exporting. The File Manager DB2 component provides an editor for DB2 tables and views (in either read-only or update mode), encompassing all the usual insert, delete, and update functionality that you typically require in a database application.

The File Manager DB2 component provides full tables, views, or the results of a limited SQL query that you have customized yourself, with the ability to save the query that you made, in a File Manager template.

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

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

You can also generate batch job control language (JCL) for the most commonly used DB2 utilities to save time.

IMS component

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

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

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

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

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

CICS component

The File Manager for CICS feature allows you to manage or query data in CICS VSAM file resources, transient data queues, and temporary data queues, allowing access to CICS resources under a CICS transaction. The CICS resources supported for view, edit, and certain File Manager utilities (such as data create, copy, and compare) are VSAM files, temporary storage queues, and transient data queues.

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

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

Another benefit of File Manager for CICS is that the majority of File Manager-related processing happens offline to the CICS task, so there is little risk of adversely affecting other CICS users, even when, for example, copying an entire file somewhere else.

File Manager for CICS maintains the same look and set of editor commands that you use in an ISPF environment. It is not necessary to log off of your CICS environment to log on to your ISPF environment if you only want to use the File Manager Base or IMS component.

6.9 Cross-platform utilization

The three PD Tools that are integrated with Rational Developer (Fault Analyzer, Debug Tool, and File Manager) complement each other by providing key ingredients to help you resolve a problem. After a job is run and an ABEND has been detected, Fault Analyzer provides key information about the ABEND in the Main Report section of the Fault Summary. The Fault Analyzer synopsis gives an explanation of what happened to cause the ABEND (Figure 6-31).

```
Module SAM2, CSECT SAM2, offset X'39E': Abend S0C7 (Data Exception)
Synopsis

I B M   F A U L T   A N A L Y Z E R   S Y N O P S I S


A system abend 0C7 occurred in module SAM2 CSECT SAM2 at offset X'39E'.

A program-interruption code 0007 (Data Exception) is associated with this abend
and indicates that:

  A decimal digit or sign was invalid.

The abend was caused by machine instruction FA442005301E (ADD DECIMAL).
```

Figure 6-31 Fault Analyzer Synopsis

You can use this information in the Debug perspective to locate the variable that is causing issues. You can set up breakpoints and step through the program in the Debug perspective. When you reach the area that is causing problems, Debug informs you that an event has occurred (Figure 6-32).

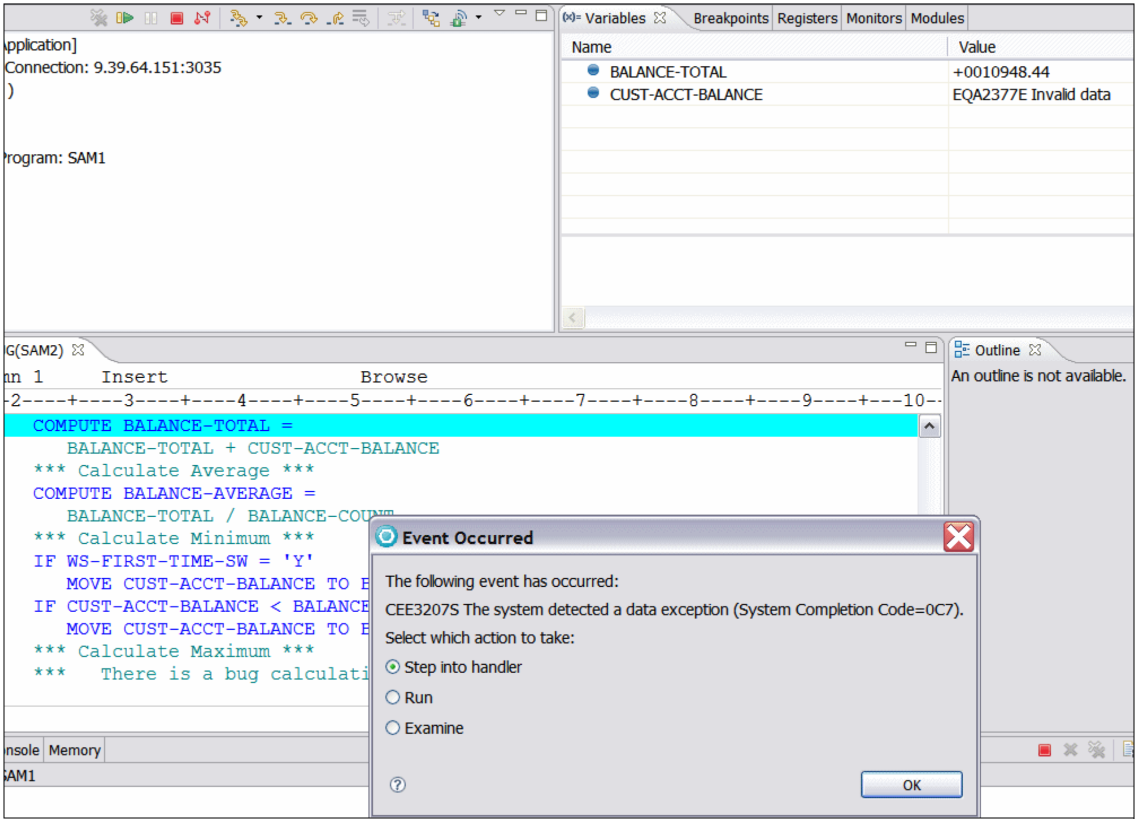


Figure 6-32 Event notification

When the variable name is found, we can begin looking at the data by going back to the z/OS perspective and selecting the data set that holds the data. You can use File Manager to look at the data. After applying the correct copybook, we can open the data set in File Manager. Double-clicking the tab for the editor of the file, we can see our file in the full window (Figure 6-33), which allows us to see a lot of data.

For those individuals familiar with the green screen version of File Manager, note that you can see both the table mode and the single mode on the same window. Selecting any record in the table section activates the single record section and synchronizes the record. It is easy to see that the record containing the name of `Turner, Paige` has bad data, which is indicated by the red asterisks in the account balance column. To fix the data, simply highlight the column, and type over the asterisks with a valid number (Figure 6-33).

F01309.far | DDS0200.ADLAB.SAMFILE

Template Associated: DDS0200.ADLAB.COPYLIB(CUSTMAST)

C...	F	NAME	ACCT-BA...	ORDERS...	ADDR	CITY	S.	CO
03115	A	Graham, Holly	254.53	1	3100 Oaktree Ct	Raleigh	NC	US
05580	A	Moore, Adeline	498.95	3	4700 S. Syracuse	Denver	CO	US
06075	A	Dubree, Dustin	192.98	1	9229 Delegate's Row	Indianapolis	IN	US
06927	A	Buchs, Jillian	99.99	0	41 Avendale Drive	Carlisle	PA	US
07008	A	Houston, Roger	296.97	1	4111 Northside PkWay	Atlanta	GA	US
07025	A	Marx, Audrey	450.51	2	90 South Cascade	Boulder	CO	US
11204	A	Ness, Luke	513.06	3	5166 Sprinkle Road	Portage	MI	US
11544	A	Graf, Polly	244.42	1	1551 S. Washington	Dowagiac	MI	US
12689	A	Boyd, Luke A.	86.88	0	221 Yale Rd	South Perth		Au
13062	A	Turner, Paige	***********	4	3039 Cornwallis Road	Boone	NC	US
13295	A	Well, Alice	731.26	4	125 Nicholson Rd	North Perth		Au

Figure 6-33 Data error

6.10 PD Tool support for languages and run times

PD Tools provide support for a variety of languages and run times.

6.10.1 IBM Debug Tool for z/OS

Debug Tool for z/OS provides debugging capability for applications running in a variety of environments, such as IBM CICS, IBM IMS, IBM DB2 stored procedures, and IBM UNIX System Services. To meet the requirements of IBM DB2 Universal Database for z/OS, Versions 8 and 9, and IBM CICS Transaction Server for z/OS, Versions 3 and 4, Debug Tool also includes features to help you identify old OS/VS and VS COBOL II source code and to upgrade the code to IBM Enterprise COBOL.

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

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

6.10.2 Support for IBM subsystems

Debug Tool works with individual load modules, independently of the subsystem in which they execute.

6.10.3 IBM Fault Analyzer for z/OS

Fault Analyzer for z/OS provides the information that you require to determine the cause and to assist with the resolution of application and subsystem failures. You can use this tool to assist in composite-application ABEND analysis. It helps you repair failures quickly by gathering information about an application and its environment at the time of failure.

CICS

Fault Analyzer functions are available within the CICS transaction-based environment in a manner that is consistent with the ISPF usage. This capability permits you to review and analyze fault entries in history files without the necessity for a Time Sharing Option (TSO) logon.

Fault Analyzer provides general information about a CICS region along with detail of control blocks, transaction storage, the last screen buffer, the trace table, and an explanation of the CICS ABEND code.

The ability of Fault Analyzer to detect and analyze dump formatting and storage violations is particularly helpful for system programmers who have to debug CICS system problems. Fault Analyzer provides options to speed up processing of duplicate dumps and to skip the analysis of repeated ABENDs.

DB2

Fault Analyzer provides the details of the last SQL activity, along with plan and package information.

IMS

Fault Analyzer provides general information about an IMS region, along with the details of the last Data Language/I (DL/I) call parameter list, information for all program communication blocks (PCBs) in the order of their relative PCB number, and, if available, job control block (JCB) call trace information. Fault Analyzer also provides IMS accounting information. You can eliminate the overhead of duplicate IMS faults occurring across all MVS images in a sysplex.

WebSphere MQ

Fault Analyzer provides application programming interface (API) information and return code diagnostics to assist in problem determination of WebSphere MQ applications.

6.10.4 Java support

When Fault Analyzer is invoked within a Java application, you can analyze ABENDing traditional programs, such as COBOL and PL/I.

A feature unique to the interactive component of Fault Analyzer is the ability to analyze Java-related information. The Java execution might be under WebSphere, CICS, or UNIX System Services on MVS. Typically, the environment is Java calling existing programs. See the *Fault Analyzer for z/OS V8R1 User's Guide and Reference*, SC19-1253-05, for instructions to perform the following tasks:

- Setting options for Java analysis
- Selecting a Java dump data set for analysis
- Displaying the resulting Java information in the interactive report
- Creating a history file entry for the analyzed dump data set

6.10.5 IBM File Manager for z/OS

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

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

CICS

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

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

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

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

File Manager for CICS maintains the same look and set of editor commands with which you are familiar in an ISPF environment. Logging off of your CICS environment to log on to your ISPF environment is unnecessary if you only want to use the File Manager Base or IMS component.

DB2

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

Database management abilities, such as creating and dropping objects, copying data within DB2, handling privileges, and importing and exporting, and an editor for DB2 tables and views (in either read-only or update mode) encompass all the usual insert, delete, and update functionality that you typically require in a database application.

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

If you write SQL to be use in your applications, you can refine and test an SQL query with the File Manager prototyping and analysis feature, which includes an explanations feature.

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

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

IMS

With File Manager's IMS component, accessing data that is stored in an IMS database also becomes easy. Although the database storage concepts differ from DB2, File Manager delivers a similar look when viewing or editing your data, and creating and saving customized queries is as easy as it is with the other components.

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

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

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

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

WebSphere MQ

Within File Manager, you can access WebSphere MQ queues on the local z/OS system where File Manager is running. From the Primary Option Menu panel, select **option 9 WebSphere MQ**. File Manager then displays the WebSphere MQ Functions panel. From this panel, you can list managers and queues, view a WebSphere MQ queue, and edit a WebSphere MQ queue.

6.10.6 Java support

File Manager can access an HFS file as a simulated QSAM/basic sequential access method (BSAM) file. Therefore, at the I/O level, the file appears as a single-volume, physical, sequential data set that resides on DASD.

7

CICS Explorer

In this chapter, we look at CICS Explorer and several of the tools that are available as plug-ins to CICS Explorer. These tools can assist you with your CICS development, maintenance, and support.

7.1 CICS Explorer

Prior to CICS Explorer, performing each CICS systems programming task required the use of various disconnected interfaces. These applications were often Interactive System Productivity Facility (ISPF)-based panels that were written to perform a specific function. Each application required custom knowledge to operate, and if the completion of a specific task crossed application boundaries, the user had to switch back and forth between the applications, which each had its own commands and user interfaces.

With CICS Explorer, CICS now gives users a much more powerful tooling environment to help experienced, skilled IT staff to be more productive and that provides a more intuitive experience for less experienced team members. It also plays a vital role in establishing and handing over best practices to the next generation of IT staff. Figure 7-1 shows how CICS Explorer can simplify the day-to-day operations of a large CICS environment.

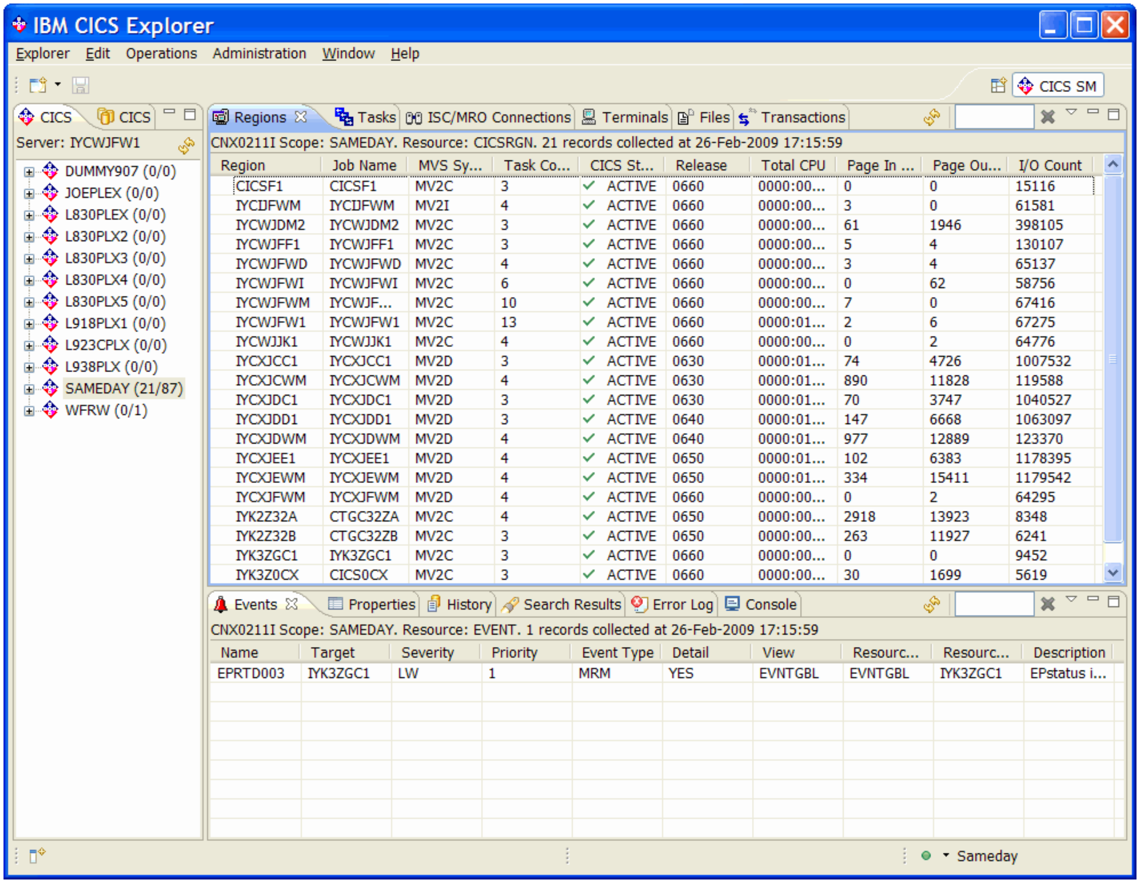


Figure 7-1 CICS Explorer

As a system management tool, CICS Explorer is designed to provide an intuitive way to manage a single CICS region, a collection of regions within an IBM CICSPlex®, or a collection of CICSPlexes. Experienced system programmers can quickly identify and tune their systems, and newer users are guided through the intuitive interface to learn how to perform standard tasks. Having a consolidated approach to system management reduces the number of interfaces that are required by CICS system administrators to learn to become effective, which helps to make the effort of attracting and training new developers and system programmers easier and less expensive.

Many of the tools now have plug-ins to CICS Explorer. See Figure 7-2 for details about all the CICS Explorer plug-ins that are currently available.

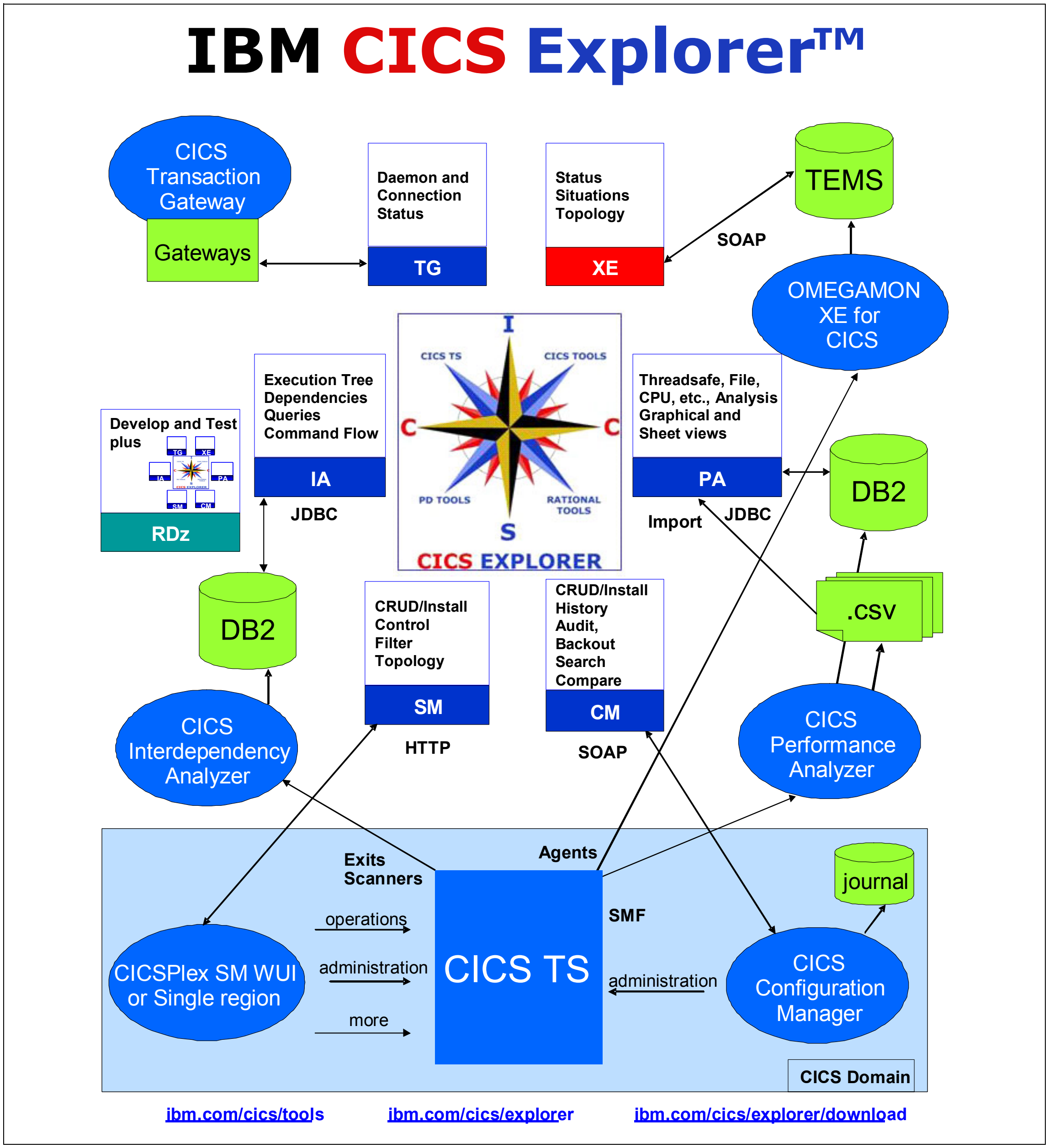


Figure 7-2 CICS Explorer and interfaces

7.2 Terms and concepts

CICS Explorer is an Eclipse rich client platform (RCP)-based GUI application that runs on the Microsoft Windows and Linux operating systems. The user interface to the applications is through the *workspace*, which is a graphical multi-window environment that is built into Eclipse. The Eclipse platform is structured as subsystems that are implemented in one or more plug-ins. The subsystems are built on top of a small runtime engine.

The workspace is a single window that consists of views and perspectives, and it can contain other elements, such as editors, menus, and the help system. CICS Explorer uses a subset of the Eclipse plug-ins to build its workspace.

CICS Explorer runs completely stand-alone, and no prior installation of Eclipse is required; however, it is necessary to understand several of the concepts from Eclipse to make full use of all that CICS Explorer offers.

7.3 Installing CICS Explorer on the z/OS server

In this section, we show the steps that are required to implement CICS Explorer for individuals who use the single system setup and for individuals who plan to use CICSPlex System Manager (SM) to take advantage of CICS Explorer.

7.3.1 Using CICS Explorer with CICS TS V4.1

CICS Transaction Server (TS) V4.1 introduces a new facility called the CICS Management Client Interface (CMCI). This new interface is exploited by CICS Explorer to provide create, read, update, and delete capabilities.

In this section, we tell you how to use CICS Explorer with CICS TS V4.1.

7.3.2 CICS Explorer can operate in two ways

CICS Explorer operates in two ways:

- Directly with CICS
- Directly with CICSPlex SM

Operating CICS Explorer directly with CICS

Smaller scale users can directly exploit CICS Explorer individual CICS regions, which provide access to operational capabilities and resource definitions through the CICS system definition data set (CSD), as shown in Figure 7-3. Small enterprises can therefore benefit from a modern management interface and integration into the CICS tools. Exploitation is a matter of minutes requiring minimal setup, with CMCI initializing the first time that a request is made to the region. The only requirements to gain access are a TCP/IP service (TCPIPSERVICE) definition and the addition of the appropriate library.

This type of usage scenario is described as a *System-Managed Single Server* (SMSS). This term is also sometimes referred to as *single server*.

CICS Explorer lets you work with the resources on a single system. You cannot use the SMSS capabilities of CICS if CICS is managed by CICSPlex SM.

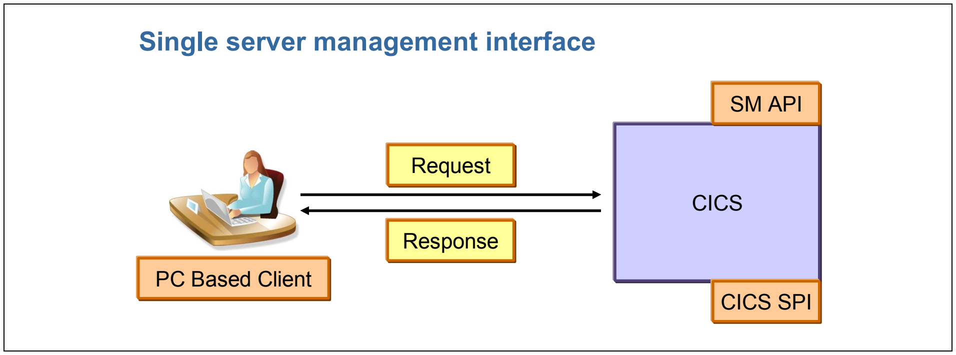


Figure 7-3 Directly with CICS

If you have multiple CICSs, each system requires a TCP/IP connection, and you need to connect CICS Explorer to each CICS in turn to work with it.

Operating CICS Explorer directly with CICSPlex SM

For those individuals who exploit the extensive capabilities that CICSPlex SM provides, you can use CICS Explorer to access the CICSPlex SM environment, as shown in Figure 7-4. In contrast to the previous support pack edition, this interface now gives you the ability to take actions on resources. Establishing access is a simple set of actions, even for a first-time CICSPlex SM user. For existing CICSPlex SM users, access is virtually immediate.

If the system is managed by CICSPlex SM, you cannot use the CICS SMSS or single server capabilities.

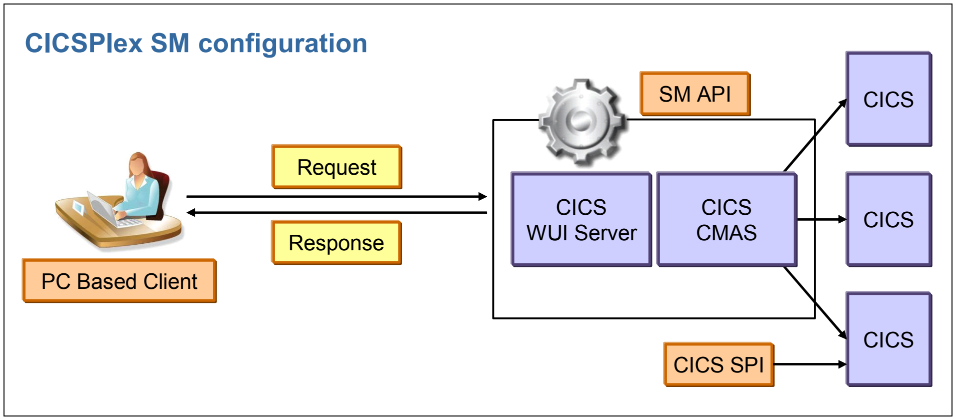


Figure 7-4 CICSPlex SM configuration

As users migrate to the advanced facilities of CICSPlex SM, the CICS Explorer experience expands into this new set of capabilities, thus eliminating the relearning curve that is often experienced when introducing new management capabilities.

7.3.3 Setting up single server access

Follow these steps to set up a single server access:

1. Copy the IBM-supplied group DFH$WU to a user-defined group. You need one group for each CICS system that you want to manage.
2. In your user-defined group, update the TCPIPSERVICE definition DFH$WUTC and the port number (each CICS requires a unique value for the TCP/IP Stack being used), and if the CICS is running with security (SEC=`YES` in the SIT), change Authenticate to `Basic`.
3. Install the user-defined group (URIMAP definition DFH$WUUR is installed alongside the DFH$WUTC.TCPIPSERVICE).
4. Add the CICSTS SEYULOAD data set to the CICS DFHRPL concatenation and the CICS TS SEYUAUTH data set to the STEPLIB concatenation in the CICS job control language (JCL). (Make sure that the CICS TS SEYUAUTH data set is authorized program facility (APF)-authorized).

Figure 7-5 on page 365 shows the CICS Explorer connection to a single server. The connection type to specify is **CICS Management Interface**. The server address to use is the IP address of the TCP/IP stack that CICS will use. The port number corresponds to the port number that is specified on the DFH$WUTC tcpipservice that you installed.

The interface initializes when the CICS region receives the first request. Example 7-1 shows a JCL that needs updating. Example 7-2 shows an updated JCL.

Example 7-1 Updating the CICS JCL (before update)

```
//CICS    EXEC PGM=DFHSIP,REGION=240M,
//             PARM=('SIT=6$',
//             'START=AUTO,SI')
//SYSIN    DD *
GRPLIST=(DFHLIST,userlist1,userlist2),
APPLID=CICSHTH1,
.END
//STEPLIB  DD  DSN=CICSTS41.CICS.SDFHAUTH,DISP=SHR
//                  DD  DSN=CICSTS41.CICS.SDFJAUTH,DISP=SHR
//DFHRPL   DD DSN=CICSTS41.CICS.SDFHLOAD,DISP=SHR
//                  DD  DSN=your.prog.library,DISP=SHR
//                  DD  DSN=your.table.library,DISP=SHR
//DFHCSD   DD DSN=CICSTS41.CICS.DFHCSD,DISP=SHR
…
```

Example 7-2 Updated JCL

```
//CICS    EXEC PGM=DFHSIP,REGION=0K,
//             PARM=('SIT=6$',
//             'START=AUTO,SI')
//SYSIN    DD *
GRPLIST=(DFHLIST,userlist1,userlist2),
APPLID=CICSHTH1,
.END
//STEPLIB  DD  DSN=CICSTS41.CPSM.SEYUAUTH,DISP=SHR
//                  DD  DSN=CICSTS41.CICS.SDFHAUTH,DISP=SHR
//                  DD  DSN=CICSTS41.CICS.SDFJAUTH,DISP=SHR
//DFHRPL   DD DSN=CICSTS41.CPSM.SEYULOAD.DISP=SHR
//                 DD  DSN=CICSTS41.CICS.SDFHLOAD,DISP=SHR
//                  DD  DSN=your.prog.library,DISP=SHR
//                  DD  DSN=your.table.library,DISP=SHR
//DFHCSD   DD DSN=CICSTS41.CICS.DFHCSD,DISP=SHR
```

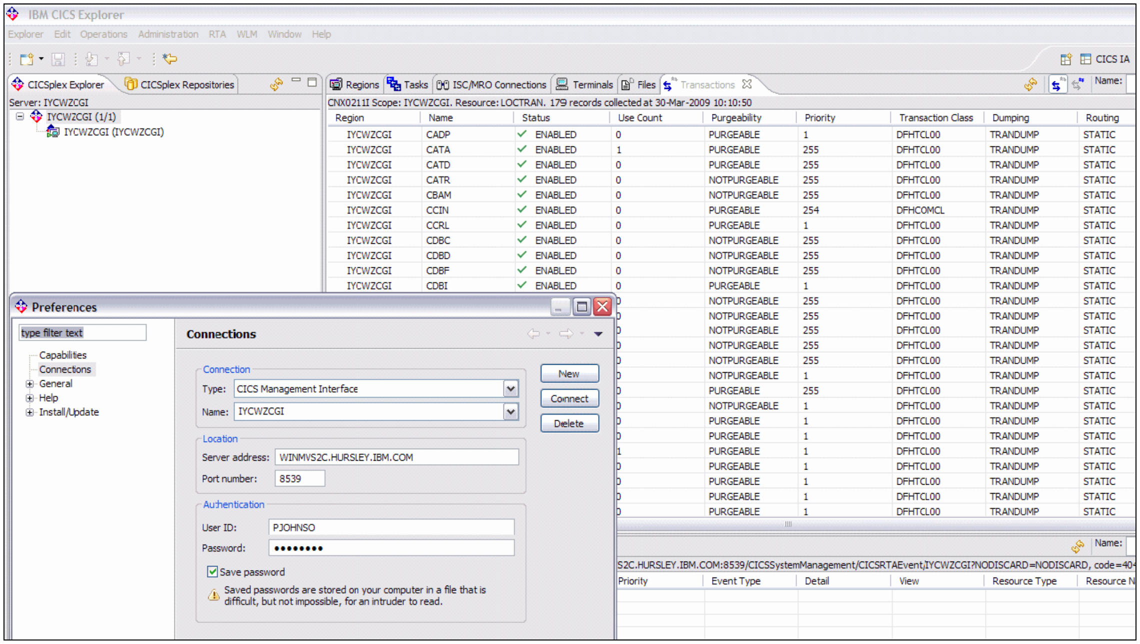


Figure 7-5 CICS Explorer connecting to a single server

7.3.4 Updating an existing CICSPlex SM setup for CICS Explorer

If you are already a CICSPlex SM user, to enable the CICS Management Client interface in your CICSPlex SM web user interface (WUI) server, update your WUI system parameters to include the new CMCIPORT parameter. The value specified is a TCP/IP port number that must be used for the CMCI. Each WUI server that is to provide CMCI support needs a unique (per TCP/IP stack) port number for the CMCIPORT.

Example 7-3 shows a WUI system parameter file that needs updating, and Figure 7-4 shows an updated system parameter file.

Example 7-3 Before updating the WUI system parameters file

```
//EYUWUI   DD *
 TCPIPHOSTNAME(MYHOST.EXAMPLE.COM)
 TCPIPPORT(60666)
 DEFAULTCONTEXT(MYPLEX)
 DEFAULTSCOPE(MYSCOPE)
 DEFAULTCMASCTXT(MYCMAS)
 AUTOIMPORTDSN(CTS41.CPSM.SEYUVIEW)
 AUTOIMPORTMEM(EYUEA*)
//
```

Example 7-4 After updating the WUI system parameters file

```
//EYUWUI   DD *
 TCPIPHOSTNAME(MYHOST.EXAMPLE.COM)
 TCPIPPORT(60666)
 CMCIPORT(50666)
 DEFAULTCONTEXT(MYPLEX)
 DEFAULTSCOPE(MYSCOPE)
 DEFAULTCMASCTXT(MYCMAS)
 AUTOIMPORTDSN(CTS41.CPSM.SEYUVIEW)
 AUTOIMPORTMEM(EYUEA*)
//
```

In the CICS Explorer Connections dialog, specify a connection type of **CICS Management Interface**, set the server address to the WUI TCPIPHOSTNAME parameter, and set the port number to match the newly specified CMCIPORT parameter.

7.3.5 Setting up CICSPlex SM for CICS Explorer

In this section, we describe the steps that are required to set up CICSPlex SM to take advantage of CICS Explorer.

7.3.6 The basics of setting up CICSPlex SM for CICS Explorer

To set up CICSPlex SM, you need two new CICS systems. In this book, we turn one of these systems into a CICSPlex SM address space (CMAS) and the second system into a WUI server. A CMAS is a CICS system that manages a CICSplex. It is a dedicated system; so, no other workloads can run inside it.

A CICSplex is simply a set of CICS systems that CICSPlex SM manages. You might have a number of CICSPlexes in your enterprise, but an individual CICS system can only belong to or be managed by one CICSplex.

A WUI server is a CICS system that acts as a web server for the web user interface into CICSPlex SM. If you use CICS Explorer with CICS TS V4.1, CICS Explorer uses this WUI server to acquire information about your CICS systems and resources. This WUI server is a dedicated system, so no other workloads can run inside it.

First, create two new CICS systems. Figure 7-6 shows an example of our environment. We made the following assumptions:

- The CICS systems are on a single logical partition (LPAR).
- You have downloaded the CICSPlex SM libraries already.
- You have JCL for the existing CICS regions.

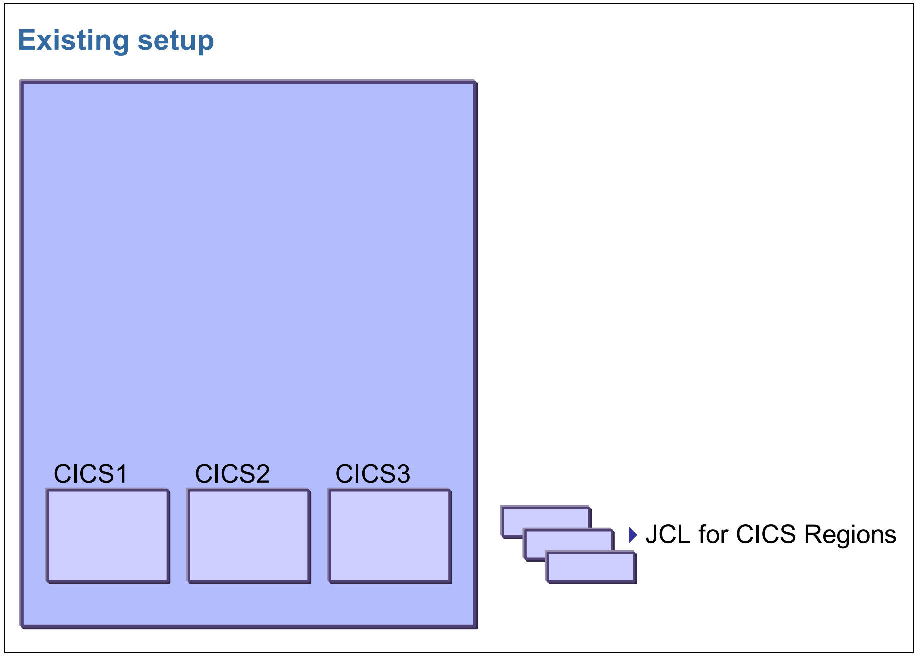


Figure 7-6 Example of existing setup

Figure 7-7 shows our environment, plus the WUI and CMAS regions.

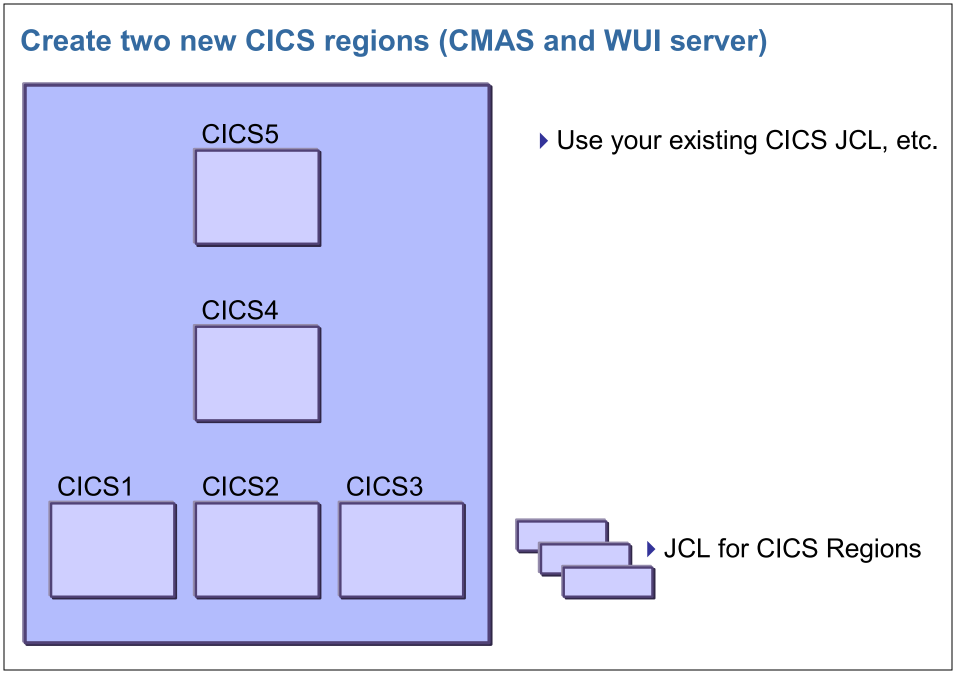


Figure 7-7 Existing setup plus WUI and CMAS regions

7.3.7 Changing a system into a CMAS

Figure 7-8 shows an example of creating a CMAS repository.

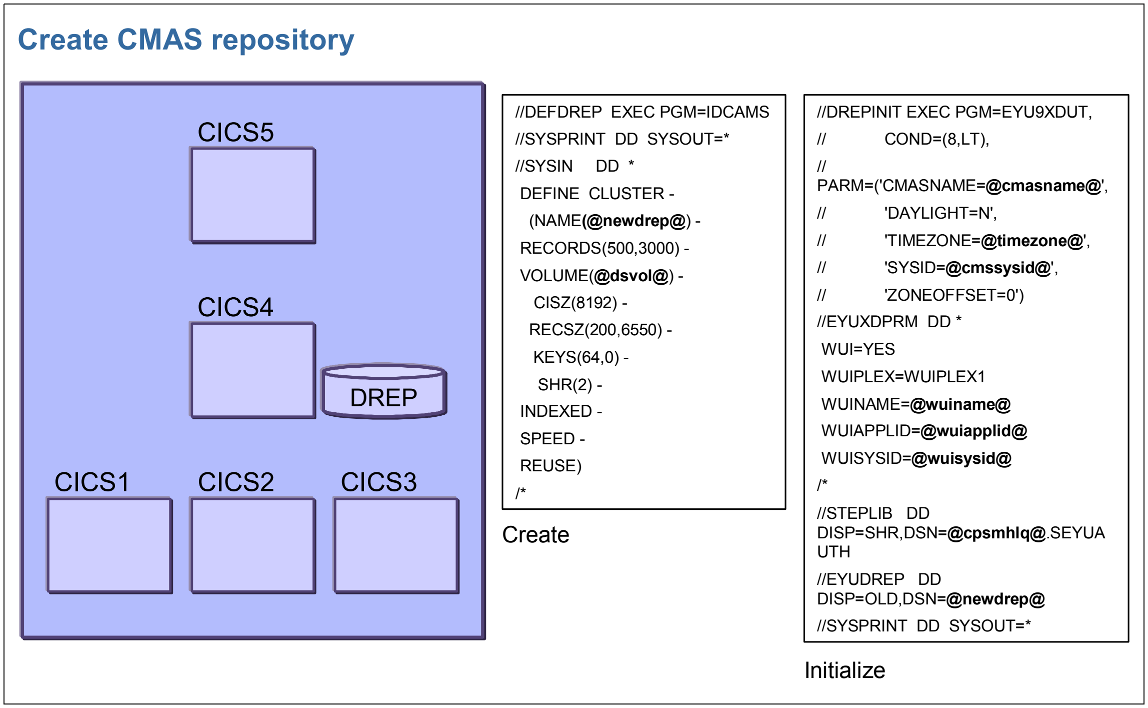


Figure 7-8 Creating CMAS repository

Creating a data repository

To change one of your systems into a CMAS, you must first create and initialize a data repository. A *data repository* is simply a Virtual Storage Access Method (VSAM) file that CICSPlex SM uses to store definitions, similar to a CICS CSD. Example 7-5 shows sample syntax for creating a data repository, and Example 7-6 provides sample JCL syntax to initialize the repository.

Example 7-5 Creating the CMAS repository

```
//DEFDREP  EXEC PGM=IDCAMS
//SYSPRINT  DD  SYSOUT=*
//SYSIN     DD  *
 DEFINE  CLUSTER -
   (NAME(@newdrep@) -
 RECORDS(500,3000) -
 VOLUME(@dsvol@) -
    CISZ(8192) -
```

```
   RECSZ(200,6550) -
    KEYS(64,0) -
     SHR(2) -
 INDEXED -
 SPEED -
 REUSE)
/*
```

Example 7-6 Initializing the CMAS repository

```
//DREPINIT EXEC PGM=EYU9XDUT,
//              COND=(8,LT),
//              PARM=('CMASNAME=@cmasname@',
//              'DAYLIGHT=N',
//              'TIMEZONE=@timezone@',
//              'SYSID=@cmssysid@',
//              'ZONEOFFSET=0')
//EYUXDPRM  DD *
 WUI=YES
 WUIPLEX=WUIPLEX1
 WUINAME=@wuiname@
 WUIAPPLID=@wuiapplid@
 WUISYSID=@wuisysid@
/*
//STEPLIB   DD  DISP=SHR,DSN=@cpsmhlq@.SEYUAUTH
//EYUDREP   DD  DISP=OLD,DSN=@newdrep@
//SYSPRINT  DD  SYSOUT=*
```

There are two variables that you must specify in the JCL for creating the repository (see Example 7-5 on page 369):

- The data set name (@newdrep@) that you want the repository to have (make a note of this name, because you will need it later)
- The volume (@dsvol@) on which you want the data set created

Initializing the data repository

Next, we must initialize the data repository. Initializing the repository loads it with several definitions that are needed for the CICSPlex to work. Example 7-6 contains sample JCL to initialize the data repository.

You must set a number of variables in the JCL. Let us start with the simplest:

1. Where @cpsmhlq@ appears, enter in the high-level qualifier for the release of CICS TS with which you are working, for example, `CTS410.CPSM.SEYUAUTH`.
2. Populate @newdrep@, which is the name you gave to your data repository when you created it with IDCAMS, as shown in Example 7-5 on page 369.
3. Populate the @cmasname@ variable. This entry is simply a name that CICSPlex SM uses to identify the CMAS. You can use up to eight characters (A-Z and 0-9) for the name. You might want to use the APPLID of the system as a name for simplicity. Make a note of the CMAS name, because you will need it later.
4. Set @timezone@ to the letter `A` to use your local time zone.
5. Set @cmssysid@, which is the system ID of the CICS system that you are turning into a CMAS.

   Find the SYSIDNT SIT parameter in your CICS JCL, and copy its value into this variable. If you do not have a SYSIDNT SIT parameter, create a four character identifier. You can use the last four characters of the system's APPLID for this value. The value must be unique.

The next set of variables helps to define a CICSplex for your WUI server and tells the CMAS about the CICS system that you are using as a WUI server:

- @wuiapplid@ is the APPLID of the CICS system that you are turning into a WUI server. *This system is not the same system that you are turning into a CMAS.* In your CICS JCL, find the APPLID SIT parameter, and copy its value into the @wuiapplid@ variable.
- @wuisysid@ is the system ID of the CICS system that you are turning into a WUI server. *This system is not the same system that you are turning into a CMAS.* In your CICS JCL, find the SYSIDNT SIT parameter, and copy its value into the @wuisysid@ variable. If you do not have a SYSIDNT SIT parameter, you must create a four character identifier. You can use the last four characters of the system's APPLID for this value. This value must be unique.
- @wuiname@ is simply a name that CICSPlex SM uses to identify the WUI server. You can use up to eight characters (A-Z and 0-9) for the name. You can use the APPLID of the system as a name. Make a note of the name, because you will need it later.

> **WUIPLEX parameter:** There is a parameter called WUIPLEX that, in this example, was set to WUIPLEX1, because the WUI is managed inside of its own CICSplex. For this exercise, it is not important that you understand why this parameter is set to this value, so just use the WUIPLEX1 value.

Both the data repository creation and initialization can be JOBSTEPS inside a single JOB.

When the JCL is submitted, you get a return code of 0, which means that your data repository data set has been created and initialized.

Modifying the CICS JCL

Now, we must modify the JCL of the CICS system that we are turning into a CMAS. Example 7-7 shows a CMAS JCL before modification. Example 7-8 shows the JCL after modification.

Example 7-7 Before updating CMAS JCL

```
//CICS    EXEC PGM=DFHSIP,REGION=240M,
//             PARM=('SIT=6$',
//             'START=AUTO,SI')
//SYSIN    DD *
GRPLIST=(DFHLIST,userlist1,userlist2),
APPLID=CICSHTH1,
.END
//STEPLIB  DD  DSN=CICSTS41.CICS.SDFHAUTH,DISP=SHR
//                  DD  DSN=CICSTS41.CICS.SDFJAUTH,DISP=SHR
//DFHRPL   DD DSN=CICSTS41.CICS.SDFHLOAD,DISP=SHR
//                  DD  DSN=your.prog.library,DISP=SHR
//                  DD  DSN=your.table.library,DISP=SHR
//DFHCSD   DD DSN=CICSTS41.CICS.DFHCSD,DISP=SHR
…
```

Example 7-8 After updating CMAS JCL

```
//CICS    EXEC PGM=EYU9XECS,REGION=0K,
//             PARM=('SIT=6$',
//             'START=AUTO,SI')
//SYSIN    DD *
GRPLIST=(DFHLIST,userlist1,userlist2),
APPLID=CICSHTH1,
MXT=300,
CPSMCONN=CMAS,
EDSALIMIT=100M,
ICVR=20000,
SEC=NO,         You can switch it on later
SPOOL=YES,
.END
//STEPLIB  DD  DSN=@cpsmhlq@.SEYUAUTH,DISP=SHR
//                  DD  DSN=CICSTS41.CICS.SDFHAUTH,DISP=SHR
```

```
//                  DD  DSN=CICSTS41.CICS.SDFJAUTH,DISP=SHR
//DFHRPL   DD DSN=@cpsmhlq@.SEYULOAD.DISP=SHR
//                 DD  DSN=CICSTS41.CICS.SDFHLOAD,DISP=SHR
//                  DD  DSN=your.prog.library,DISP=SHR
//                  DD  DSN=your.table.library,DISP=SHR
//DFHCSD   DD DSN=CICSTS41.CICS.DFHCSD,DISP=SHR
//EYUDREP DD DISP=SHR,DSN=@newdrep@
//EYUPARM DD *
 NAME(@cmasname@)
 SEC(NO)
/*
```

Follow these steps to modify the JCL:

1. In the JOBSTEP for the CICS system, change the program from DFHSIP to `EYU9XECS`. Also, add `REGION=0K` to this line.
2. Add the following data set to your STEPLIB (the variables that you use depend on the release of CICS TS that you use):

   @cpsmhlq@.SEYUAUTH
3. Add the following data set to DFHRPL:

   @cpsmhlq@.SEYULOAD
4. Add the following SIT parameter:

   CPSMCONN=CMAS
5. Add or change the MXT SIT parameter, which is the minimum MAXTASKs value for a CMAS (for a basic setup, `300` is an appropriate value):

   MXT=300
6. Change the EDSALIMIT SIT parameter, which is the minimum dynamic storage area (DSA) size for a CMAS (for a basic setup, `100M` is an appropriate value):

   EDSALIMIT=100M
7. Add or change the ICVR SIT parameter:

   ICVR=20000
8. Add or change the SEC SIT parameter. Initially, we will run with security turned off. You can run with security turned on at a later date:

   SEC=NO
9. Add or change the SPOOL SIT parameter:

   SPOOL=YES

10. Change the START SIT parameter:

    START=AUTO

    Note that you might want to force a COLD/INITIAL start on the first startup, but an AUTO restart is acceptable after that.

11. Add the following data definition (DD) card to the JCL, where @newdrep@ is the name of the data repository that you created earlier:

    //EYUDREP DD DISP=SHR,DSN=@newdrep@

12. Add the following DD card:

    //EYUPARM DD *

        NAME(@cmasname@)

        SEC(NO)

    /*

> **Important:** This example assumes that the IBM-supplied SIT 6$ is used. If you use an installation-defined SIT, you might need to modify other values that are not shown here. Otherwise, incorrect results can occur.

Example 7-8 on page 372 illustrates all of these changes. You are now ready to start your CMAS.

Starting the CMAS

Submit the JCL that you have modified. If the CMAS starts successfully, in the logs, you see the following messages:

```
EYUXL0007I IYK3ZDD1 CMAS Phase III initialization complete.
EYUXL0007I IYK3ZDD1 CMAS Phase IV initialization complete.
EYUXL0010I IYK3ZDD1 CMAS initialization complete.
```

If CMAS initialization does not complete, go back to make sure that all the names, APPLIDs, and system IDs that you defined are correct.

7.3.8 Changing a system into a WUI server

The first step in creating a WUI server is to create a WUI repository, as shown in Figure 7-9.

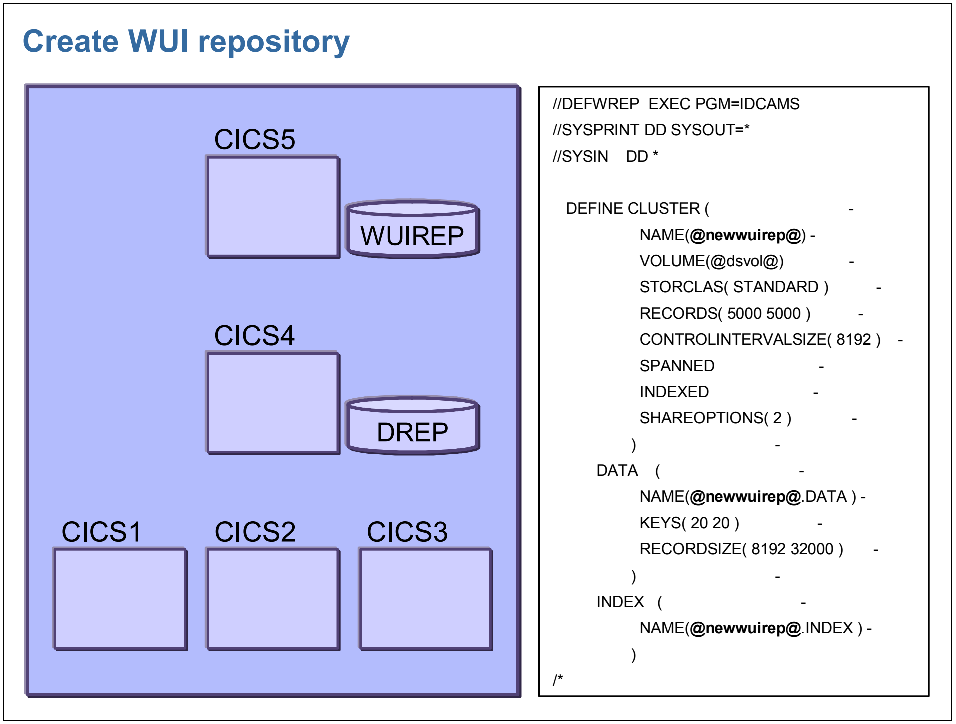


Figure 7-9 Creating a WUI repository

The WUI repository is simply a VSAM data set that holds all the views and menus that the WUI serves to a browser. Example 7-9 contains sample JCL to create a WUI repository.

Example 7-9 Creating a WUI repository

```
//DEFWREP  EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *

   DEFINE CLUSTER (                                -
                    NAME(@newwuirep@) -
                    VOLUME(@dsvol@)               -
                    STORCLAS( STANDARD )           -
                    RECORDS( 5000 5000 )           -
```

```
                    CONTROLINTERVALSIZE( 8192 )    -
                    SPANNED                        -
                    INDEXED                        -
                    SHAREOPTIONS( 2 )              -
                  )                                -
          DATA    (                                -
                    NAME(@newwuirep@.DATA ) -
                    KEYS( 20 20 )                  -
                    RECORDSIZE( 8192 32000 )       -
                  )                                -
          INDEX   (                                -
                    NAME(@newwuirep@.INDEX ) -
                  )
/*
```

The @newwuirep@ is the data set name that you want for the WUI repository.

We are now ready to modify our CICS system's JCL to turn it into a WUI server.

Modifying the CICS JCL

Example 7-10 shows the changes to make to the CICS JCL to make it a WUI server.

Example 7-10 Changing the CICS JCL to make it a WUI server

```
//CICS    EXEC PGM=DFHSIP,REGION=OK,
//             PARM=('SIT=6$',
//             'START=AUTO,SI')
//SYSIN    DD *
GRPLIST=(DFHLIST,userlist1,userlist2),
APPLID=CICSHTH1,
CPSMCONN=WUI
INITPARM=(EYU9VWAN='ENU1',EYU9VKEC='ENU'),
EDSALIM=100M,
MXT=120,
TCPIP=YES,
SEC=NO,         You can switch it on later
.END
/*
//STEPLIB  DD  DSN=@cpsmhlq@.SEYUAUTH,DISP=SHR
//                  DD  DSN=CICSTS41.CICS.SDFHAUTH,DISP=SHR
//                  DD  DSN=CICSTS41.CICS.SDFJAUTH,DISP=SHR
//DFHRPL   DD DSN=@cpsmhlq@.SEYULOAD.DISP=SHR
//                 DD  DSN=CICSTS41.CICS.SDFHLOAD,DISP=SHR
//                  DD  DSN=your.prog.library,DISP=SHR
```

```
//                  DD  DSN=your.table.library,DISP=SHR
//DFHCSD   DD DSN=CICSTS41.CICS.DFHCSD,DISP=SHR

//EYUWREP DD DISP=SHR,DSN=@newdrep@
//EYUPARM DD *
 NAME(@wuiname@)
 CICSPLEX(WUIPLEX1)
 CMASSYSID(@cmassysid@)
 SEC(NO)
/*
//EYUWUI DD *
 TCPIPHOSTNAME(@hostname@)
 TCPIPPORT(@portnumber@)
 DEFAULTCONTXT(@mainlex@)
 DEFAULTSCOPE(@mainplex@)
 DEFAULTCMASCTXT(@cmasname@)
 AUTOIMPORTDSN(@cpsmhlq@.SEYUVIEW)
 AUTOIMPORTMEM(EYUEA*)
 CMPIPORT=@portnumber@
/*
```

Now, we must modify the JCL of the CICS system that we are turning into a WUI server:

1. Add the following data set to your STEPLIB (the variables that are used depend on the release of CICS TS that you use):

   @cpsmhlq@.SEYUAUTH

2. Add the following data set to DFHRPL:

   @cpsmhlq@.SEYULOAD

3. Add the following SIT parameter:

   INITPARM=(EYU9VWAN='ENU1',EYU9VKEC='ENU')

4. Make sure that the TCP SIT parameter is set to `YES` and that the CICS system is capable of running as a web server.

5. Add the following DD card:

   //EYUPARM DD *

    NAME(@wuiname@)

    CICSPLEX(WUIPLEX1)

    CMASSYSID(@cmssysid@)

    SEC(NO)

> **Important:** The @wuiname@ must match the WUI name that you specified when creating the CMAS data repository. The @cmssysid@ must match the system ID of the CMAS, not the WUI server system ID.

//EYUWUI DD *

TCPIPHOSTNAME(@hostname@)

TCPIPPORT(@portnumber@)

DEFAULTCMASCTXT(@cmasname@)

DEFAULTCONTEXT(@mainplex@)

DEFAULTSCOPE(@mainplex@)

AUTOIMPORTDSN(@cpsmhlq@.SEYUVIEW)

AUTOIMPORTMEM(EYUEA*)

CMPIPORT(@cmciportnumber@)

The following definitions explain the variable values in the preceding DD card:

- @hostname@ is the TCP/IP host name for the IP stack that CICS will use. You also use this value when configuring the connection to this WUI server in CICS Explorer.
- @portnumber@ is a port number that was allocated for the use of the WUI server. The web browser users use this value.
- @cmasname@ is the name of the CMAS to which you are connecting, for example, the name of the CMAS that you created earlier.
- @mainplex@ is the name of the CICSPlex to which you adding your existing CICS systems. It can be any eight-character string (A-Z and 0-9). You create this CICSPlex later. In this example, we use a CICSPlex name of `TESTPLEX`, so in our WUIPARM, we have these values:
  - DEFAULTCONTEXT(TESTPLEX)
  - DEFAULTSCOPE(TESTPLEX)
- @cpsmhlq@ is the high-level qualifier for the release of CICS TS with which you are working, for example, `CICSTS41.CPSM.SEYUVIEW`.
- @cmciportnumber@ is the port number that is allocated for the use of the CICS Management Client interface for this WUI server. This value must differ from the @portnumber@ value. You use this value when configuring the connection to this WUI server on a CICS Management Interface connection in CICS Explorer.

Example 7-10 on page 376 illustrates all of these changes. You are now ready to start the WUI server.

Starting the WUI Server

Submit the JCL that you have just modified. If the WUI starts successfully, you see messages similar to the following messages in the system log:

```
EYUXL0030I IYK3ZDD4 ESSS connection in progress to CICSPLEX(WUIPLEX1)
for SYSID(@cmssysid@).
EYUXL0004I IYK3ZDD4 ESSS connection complete.
EYUCL0006I IYK3ZDD4 ESSS link to IYK3ZDD1 established.
EYUXL0007I IYK3ZDD4 LMAS Phase II initialization complete.
EYUNL0099I IYK3ZDD4 LMAS LRT initialization complete.
```

At the top of the EYULOG, you see messages similar to the following messages. There are additional messages after these messages, but do not worry about them, because they mostly relate to the WUI loading the views and menus that it will use:

```
EYUVS0101I IYK3ZDD4 Parameter service initialization complete.
EYUVS0200I IYK3ZDD4 Starting CICS Web Interface.
EYUVS0204I IYK3ZDD4 TCP/IP service (EYUWUI) installed successfully.
EYUVS0206I IYK3ZDD4 CICS Web Interface enabled on TCP/IP port number
@portnumber@.
EYUVS0204I IYK3ZDD4 TCP/IP service (EYUCMCIT) installed successfully.
EYUVS0209I IYK3ZDD4 URI map (EYUCMCIU) installed successfully.
EYUVS0211I IYK3ZDD4 CICS Management Client Interface enabled on TCP/IP
port number @cmciportnumber@.
EYUVS0002I IYK3ZDD4 CICSPlex SM Web User Interface initialization
complete.
EYUVS0010I IYK3ZDD4 Server connected to CMAS, SYSID(@cmssysid@).
```

If WUI initialization does not complete, go back to make sure that all the names, APPLIDs, and system IDs that you defined are correct.

You are now ready to create a CICSplex for all the CICS systems that you want to manage.

7.3.9 Creating a CICSplex to manage your systems

Now, we define a CICSplex that will be used to manage your existing CICS systems. We also define your CICS systems to the CICSplex, so that when you start those systems, CICSPlex SM knows by which CICSplex they are being managed. Example 7-11 shows sample JCL for creating the CICSplex and defining your CICS systems.

Example 7-11 Sample JCL to create the CICSplex

```
//CPSMDEFN EXEC PGM=EYU9XDBT
//*
//STEPLIB   DD   DISP=SHR,DSN=@cpsmhlq@.SEYULOAD
//          DD   DISP=SHR,DSN=@cpsmhlq@.SEYUAUTH
//*
//SYSTSPRT  DD   SYSOUT=*
//EYUTRCF   DD   SYSOUT=*
//*
//EYUCPDEF  DD   DISP=SHR,DSN=@cpsmhlq@.SEYUPARM(EYUCPDEF)
//EYUCSDEF  DD   DISP=SHR,DSN=@cpsmhlq@.SEYUPARM(EYUCSDEF)
//EYUCSGRP  DD   DISP=SHR,DSN=@cpsmhlq@.SEYUPARM(EYUCSGRP)
//*
//EYUSYSIN  DD   DDNAME=SYSIN
* Set CONTEXT to CMAS
CONTEXT @cmasname@
* Define the CICSplex
DEFINE CICSPLEX @plexname@
* Set CONTEXT to the CICSplex
CONTEXT @plexname@
* Define a CICS region
DEFINE REGION    @systemname@ -
       APPLID    @systemapplid@ -
       SYSID     @systemsysid@ -
       CMASID    @cmasname@
```

Follow these steps to create a CICSPlex that manages your systems:

1. @cpsmhlq@ is the high-level qualifier for the release of CICS TS with which you are working, for example, CICSTS41.CPSM.
2. @cmasname@ is the name of the CMAS to which you are connecting, for example, the name of the CMAS that you created earlier.
3. @plexname@ is the name of the CICSplex that is going to manage your systems. This name is any name up to eight characters long (A-Z and 0-9). For this example, `TESTPLEX` is a valid name to enter here.
4. @systemapplid@ is the APPLID of a CICS system that you want to manage in your CICSplex. Find the APPLID SIT parameter in the JCL for the system that you want to manage, and copy its value into this variable.
5. @systemsysid@ is the system ID of the CICS system that you want to manage in your CICSPlex. Find the SYSIDNT SIT parameter in your CICS JCL, and copy its value into this variable. If you do not have a SYSIDNT SIT parameter, create a four-character identifier. You can use the last four characters of the system's APPLID for this value. This value must be unique.

6. @systemname@ is simply a name that CICSPlex SM uses to identify the system. You can use up to eight characters (A-Z and 0-9) for the name. You can use the APPLID of the system as a name. Make a note of the name, because you will need it later.
7. Repeat the DEFINE REGION statement for every CICS system that you want to manage in your new CICSplex. Do not forget to change that variable to match the APPLID of each system.
8. Submit the job, which creates all your new definitions.

You are almost ready to manage your CICS systems through CICSPlex SM. The last step before starting your systems is to alter the JCL of your CICS systems to allow them to connect to CICSPlex SM.

7.3.10 Modifying your existing CICS system's JCL

Use the JCL that you have for an existing CICS system that you want managed by CICSPlex SM, and make the following modifications to it:

1. Add the following data set to your STEPLIB (the variables used depend of the release of CICS TS that you use):

   @cpsmhlq@.SEYUAUTH

2. Add the following to DFHRPL:

   @cpsmhlq@.SEYULOAD

3. Add the following SIT parameter:

   CPSMCONN=LMAS

4. Add or change the SEC SIT parameter (you can turn on security at a later date):

   SEC=NO

5. Add the following DD card:

   //EYUPARM DD *

   NAME(@cicsname@)

   CICSPLEX(@plexname@)

   CMASSYSID(@cmsysid@)

The variable@cicsname@ is the name of the CICS system that we defined in the WUI, which must match the name that is defined in the WUI, or else the system will not connect to the CICSplex. In our example, our system names are the same as our systems' APPLIDs. @plexname@ is the name of the CICSplex that we created in the WUI; however, in our example, this name was `TESTPLEX`. @cmsysid@ is the system ID of the CICS system that is acting as our CMAS.

After you make these changes, you can submit your JCL to start your managed CICS system. If the CICS system successfully connects to the CICSplex, you see the following messages in the system log:

```
EYUXL0030I IYK3ZDD4 ESSS connection in progress to CICSPLEX(DWDPLEX1)
for SYSID(DANA).
EYUXL0004I IYK3ZDD4 ESSS connection complete.
EYUCL0006I IYK3ZDD4 ESSS link to IYK3ZDD1 established.
EYUXL0007I IYK3ZDD4 LMAS Phase II initialization complete.
EYUNL0099I IYK3ZDD4 LMAS LRT initialization complete.
```

If the connection to the CICSPlex fails, check that the NAME in the EYUPARM in your JCL matches the “CICS system definition name” for the definition that you defined through the WUI.

Repeat this step for all the CICS systems that you want CICSPlex SM to manage.

7.3.11 Conclusion

We followed these steps to set up CICSplex SM:

1. Turned a CICS system into a CMAS
2. Turned a CICS system into a WUI server
3. Created a CICSplex
4. Defined the CICS systems to the CICSplex
5. Modified the existing CICS systems to allow them to connect to the CICSplex

You can now view information about your systems through CICS Explorer or the CICSPlex SM web user interface.

To use CICS Explorer, set up a new CICS Management Interface connection. The host name is the value of @hostname@ that you specified on the TCPIPHOSTNAME option of the EYUWUI DD. The port number to use is the value of @cmciportnumber@ that you specified on the CMCIPORT option of the EYUWUI DD. See Figure 7-10.

Use a URL of `http://@hostname@:@portnumber@` to test the web browser functionality. The following definitions explain the variables:

- @hostname@ is the value that you specified on the TCPIPHOSTNAME option of the EYUWUI DD
- @portnumber@ is the value of @portnumber@ that you specified on the TCPIPPORT option of the EYUWUI DD

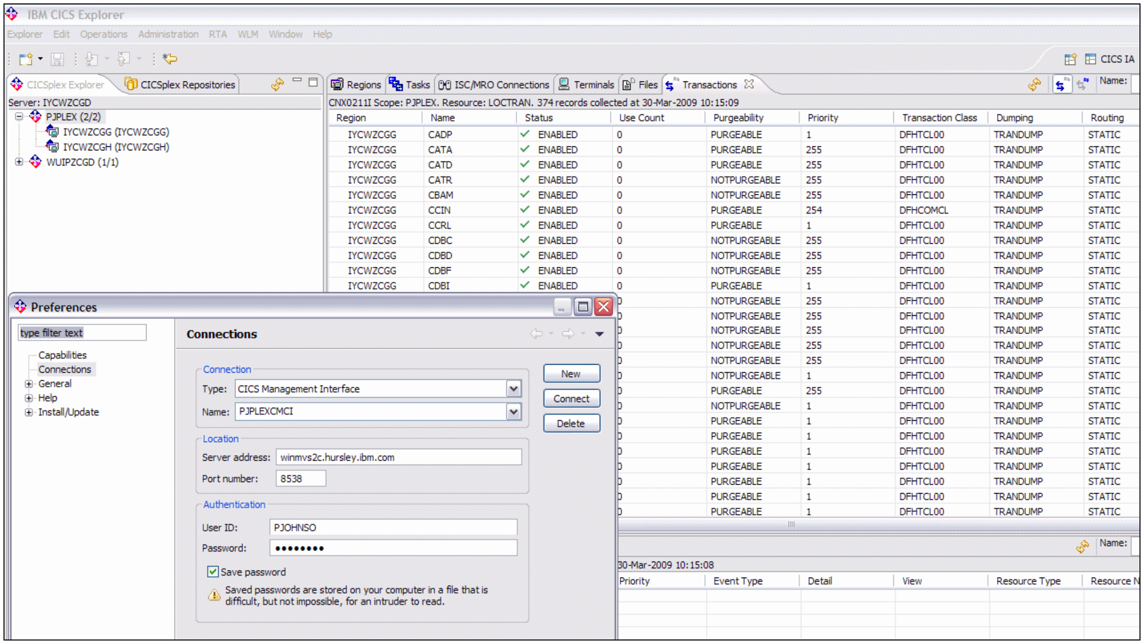


Figure 7-10 Connecting to CICS Explorer

7.4 Installing the CICS Explorer GUI

CICS Explorer offers a range of installation options. In this section, we describe the various installation options. When deciding on the best approach for installing the CICS Explorer within an organization, it is important to consider your corporate guidelines and security requirements for client applications.

You can install CICS Explorer on each user's local client machine or on a remote server for all users to access. Regardless of which option you choose, you can save a user's customized data to the user private storage area.

Both local and remote installations of CICS Explorer start with the download of the appropriate distribution for your operating system. CICS Explorer downloads as an archive file. For Microsoft Windows, this archive file is a compressed file. For Linux, this archive file is a tar file.

To download CICS Explorer, follow this link and then follow the link under Downloads:

http://www.ibm.com/software/htp/cics/explorer/

7.4.1 Installing CICS Explorer locally

Extract the distribution code of CICS Explorer using the appropriate method for your operating system, and create the following structure (Figure 7-11 reflects the Microsoft Windows view, and Figure 7-12 reflects the Linux view).

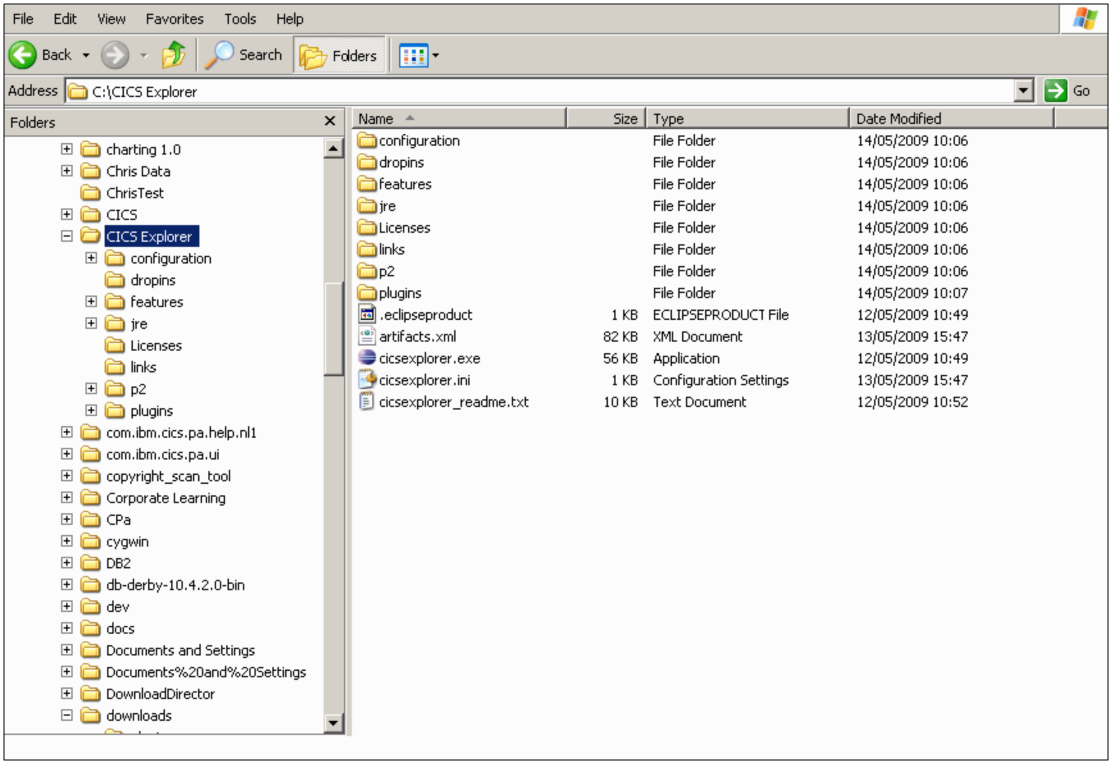


Figure 7-11 Microsoft Windows view

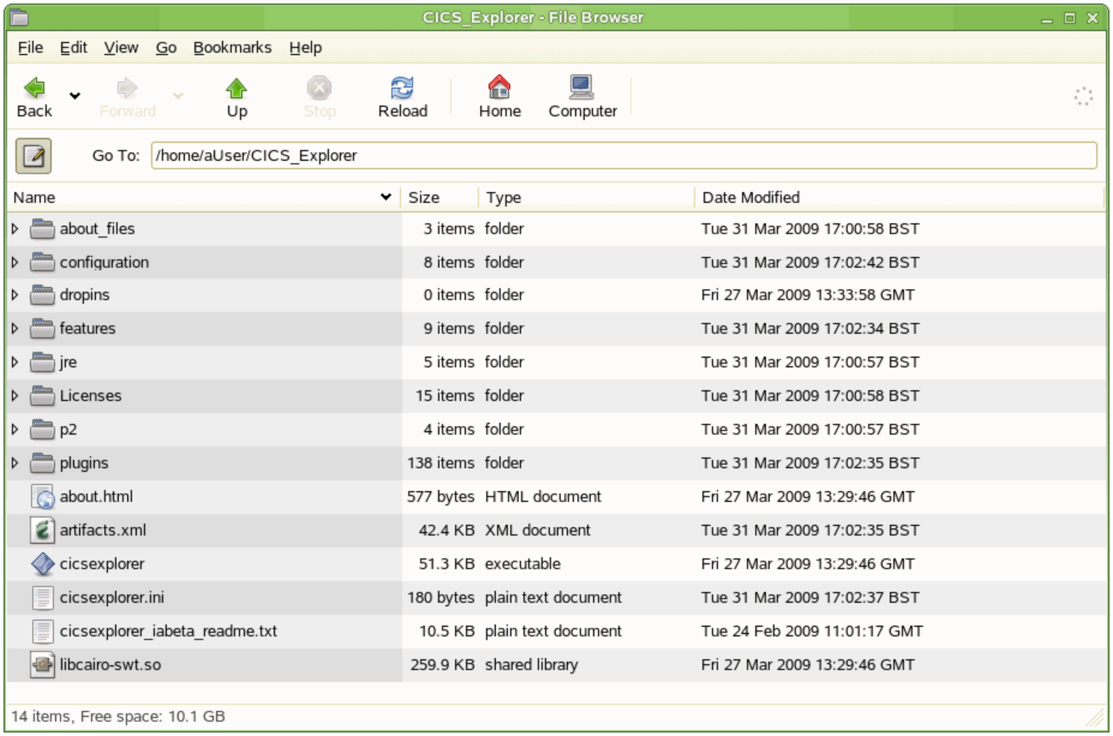


Figure 7-12 Linux view

Because no setup application is required to install CICS Explorer, we advise that you expand the archive file to a suitable location on the system's file system, for example, to `c:\program files\` on Microsoft Windows and `/opt/` on Linux.

You might choose to create a desktop shortcut to the `cicsexplorer` executable file; however, at this point, the installation is complete.

Along with the distribution code at run time, CICS Explorer creates a separate directory that is known as the *workspace* where user-specific settings and customizations are written. By default, this workspace is in the user's home directory. On Microsoft Windows, this directory is usually the `c:\Documents and Settings\<<`*`User Name`*`>>\.cicsexplorer\`, and on Linux, this directory is `/home/<<`*`User Name`*`>>/.cicsexplorer`.

> **Remember:** On certain operating systems, folder names beginning with a dot do not display, by default.

You might want to change the location of the workspace, for example, to a shared drive to be accessed from multiple locations. To make this change, you need to supply the -data option when invoking the CICS Explorer executable file. You can make this change from the command line. Or, you can create a shortcut to the file, including the -data option, for example:

```
cicsexplorer.exe -data d:\myfiles\myworkspace
```

7.4.2 Installing CICS Explorer remotely

You can install CICS Explorer on a remote server for clients to access. This approach is beneficial in that you only have a single copy of the binary to update; however, execution might be a little slower than running the client locally.

The simplest way to install CICS Explorer remotely is to install CICS Explorer on a shared drive. Each client machine can then access the shared drive. Users can then start CICS Explorer by directly running the executable file or by creating a local shortcut for the remote executable on the local machine. When CICS Explorer is run in this way, the user's workspace is written to the user's local machine, unless explicitly specified. When CICS Explorer is run from a remote drive, there might be a delay in the time that it takes for CICS Explorer to start as the binary data is copied over the network.

The benefits of running CICS Explorer in this way are that maintenance needs to be applied to only a single copy of the application binary, and this method removes the overhead of rolling the application out to every workstation. The disadvantage of running CICS Explorer in this manner is that the initial start-up time of CICS Explorer might take longer than if you run it locally.

You can run CICS Explorer in this way regardless of which operating system is used, because the remote server is merely acting as a file store. All the processing is performed on the local machine.

You can run CICS Explorer from a shared Linux server using the X Window system to display the GUI on a user's local terminal. To run CICS Explorer from a shared Linux server using the X Window system, install CICS Explorer on a Linux server in a location that is readable by logged-on users. Users can then log in to the remote system using their preferred X windowing package. After they log in, users can run CICS explorer on the remote system. When CICS Explorer is run from a remote Linux server in this manner, the CICS Explorer binary executes on the server using the server's memory and CPU resources. The user's workspace is not stored in the user's home directory and not on the local workstation.

7.5 Installing plug-ins

You download individual plug-ins from the appropriate support sites in the form of a `site.zip` file, which contains the executable code and the necessary license agreements. After you download the file on your machine, you can integrate the plug-ins into the installation by following these steps:

1. Expand the **Help** menu, and select **Software Updates** (Figure 7-13).

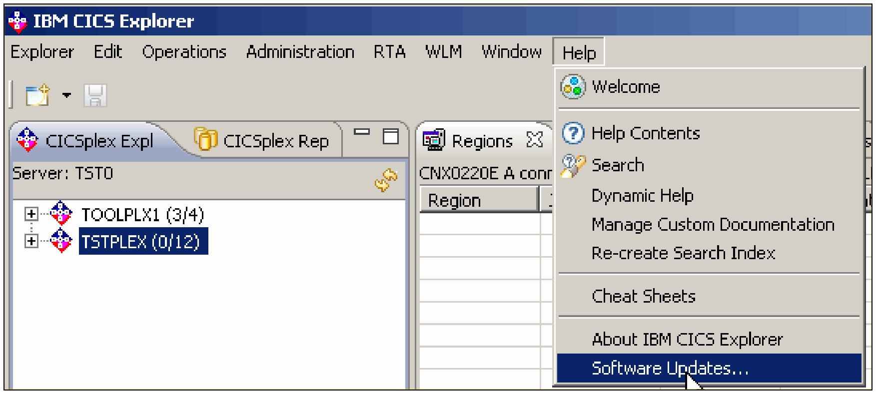


Figure 7-13 Software Updates menu

2. Select the **Available Software** tab, and click **Manage Sites** (Figure 7-14).

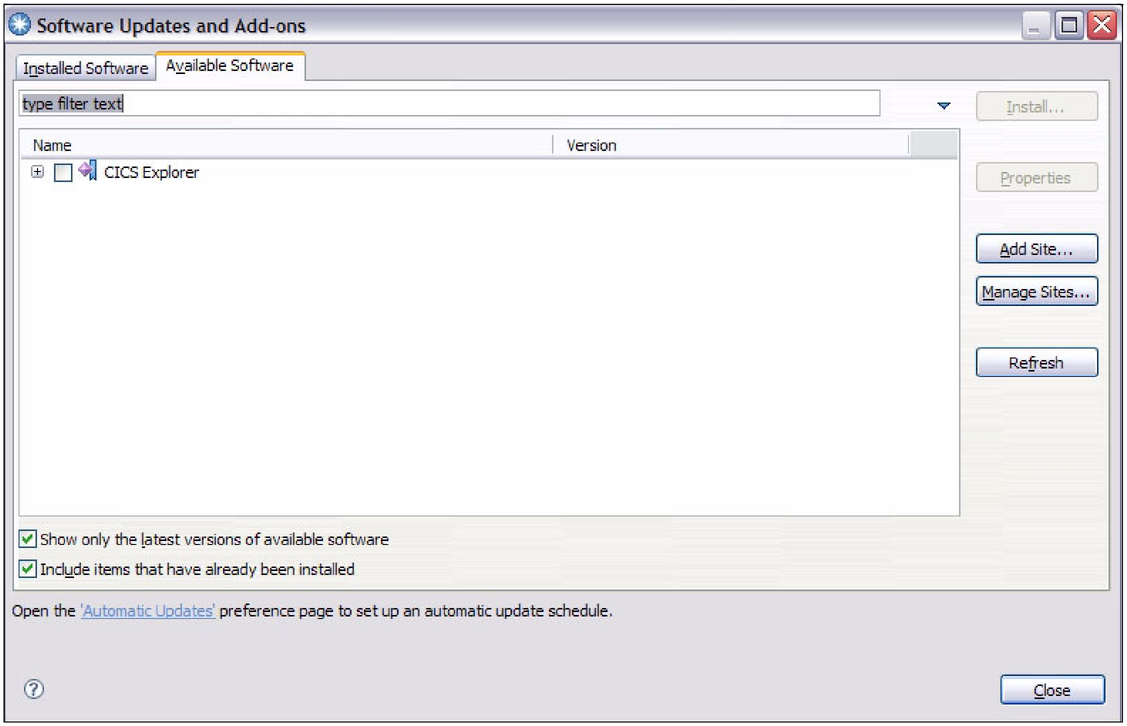


Figure 7-14 Available Software tab

3. Select **Add**. In the Add Site pop-up window, click **Archive** to select the compressed file that you downloaded (Figure 7-15).

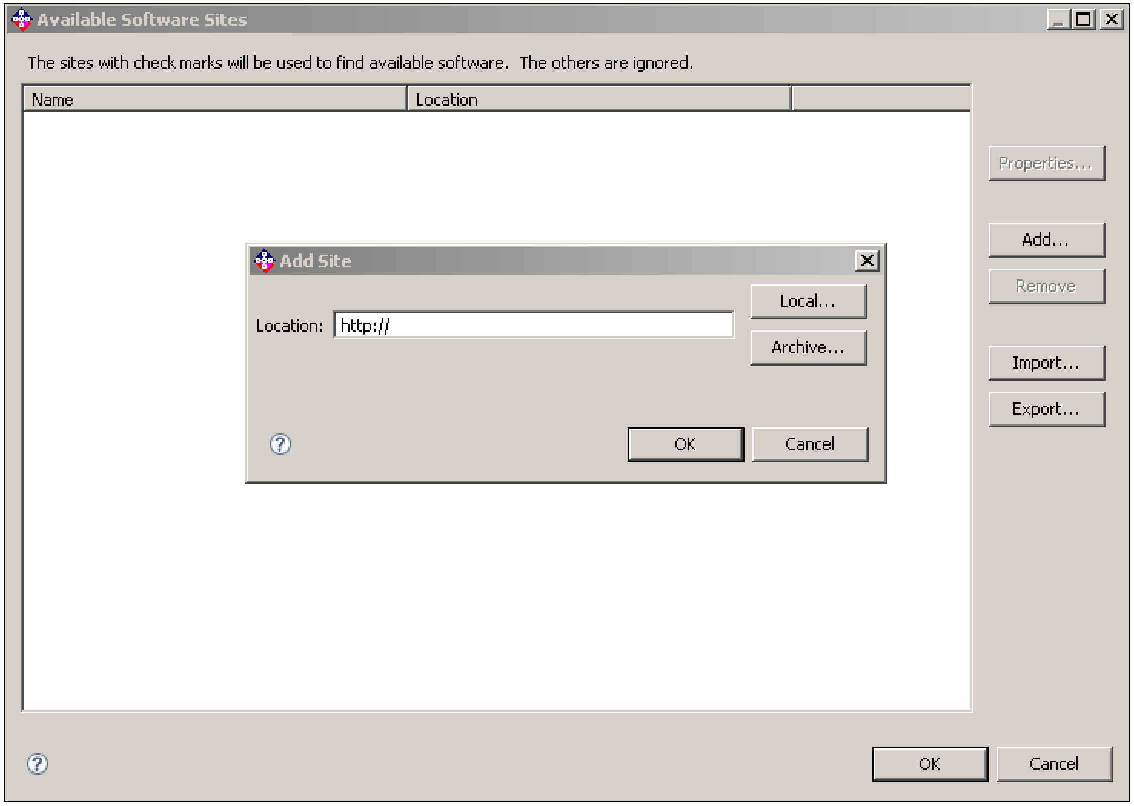


Figure 7-15 Add Site pop-up window

4. Click **OK** to close the pop-up window.
5. Click **OK** to close the Available Software Sites window.

6. In the Available Software tab, click the check box beside the plug-in that you want to install.

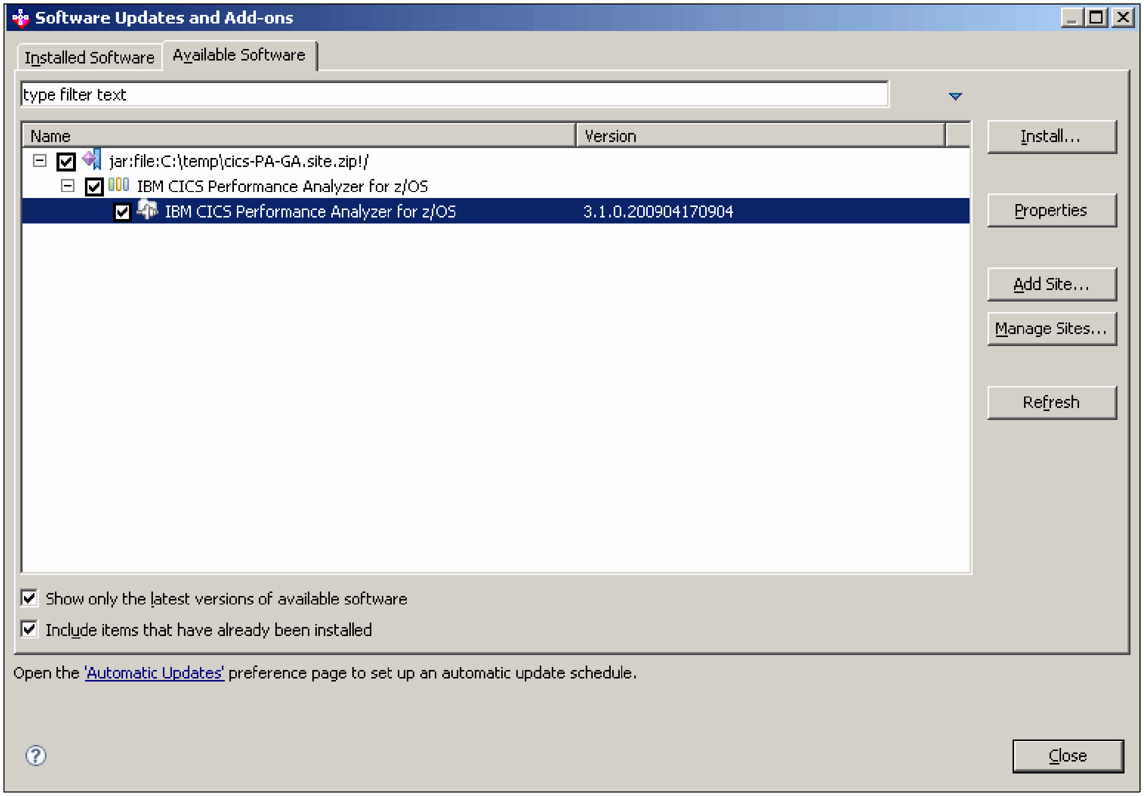


Figure 7-16 Plug-in check box

7. Click **Install**.
8. Eclipse ensures that your selection is compatible with your installation. If everything is OK, the system prompts you to accept the terms of the license agreement. Accept the terms.
9. After you have selected that you agree to the terms of the license, click **Finish**.
10. After the installation finishes, the system prompts you to restart CICS Explorer. Click **Yes**.
11. After the installation completes, the welcome window opens to show the new functionality that has been added to CICS Explorer.

See *CICS Explorer*, SG24-7778, for more details about installing CICS Explorer.

7.6 Using the CICS Explorer

With CICS TS V4.1, you can use CICS Explorer to create new resource types, such as CICS event bindings and CICS bundles.

As well as typical CICS system management and the definition and operation of regions and their installed resources, a typical CICS shop uses a number of tools for tasks, such as system configuration, debugging, and performance monitoring. You typically obtain these tools from IBM, third-party vendors, or you develop them in-house. Each tool has its own interfaces and its own method of presenting the CICS run time.

One of the driving forces behind the creation of CICS Explorer was to provide an integration platform where all the tools and applications, which a user needs to perform a business task, work together coherently in an integrated environment. The benefit of this integration is to help new users become familiar with the platform, to present a single and reusable way of accessing and manipulating CICS to promote consistency, and to allow tools that are built by IBM, vendors, and clients to be integrated within a single environment.

For example, Figure 7-17 shows how you can plug CICS Configuration Manager into CICS Explorer, and Figure 7-18 shows how you can plug CICS Interdependency Analyzer into CICS Explorer.

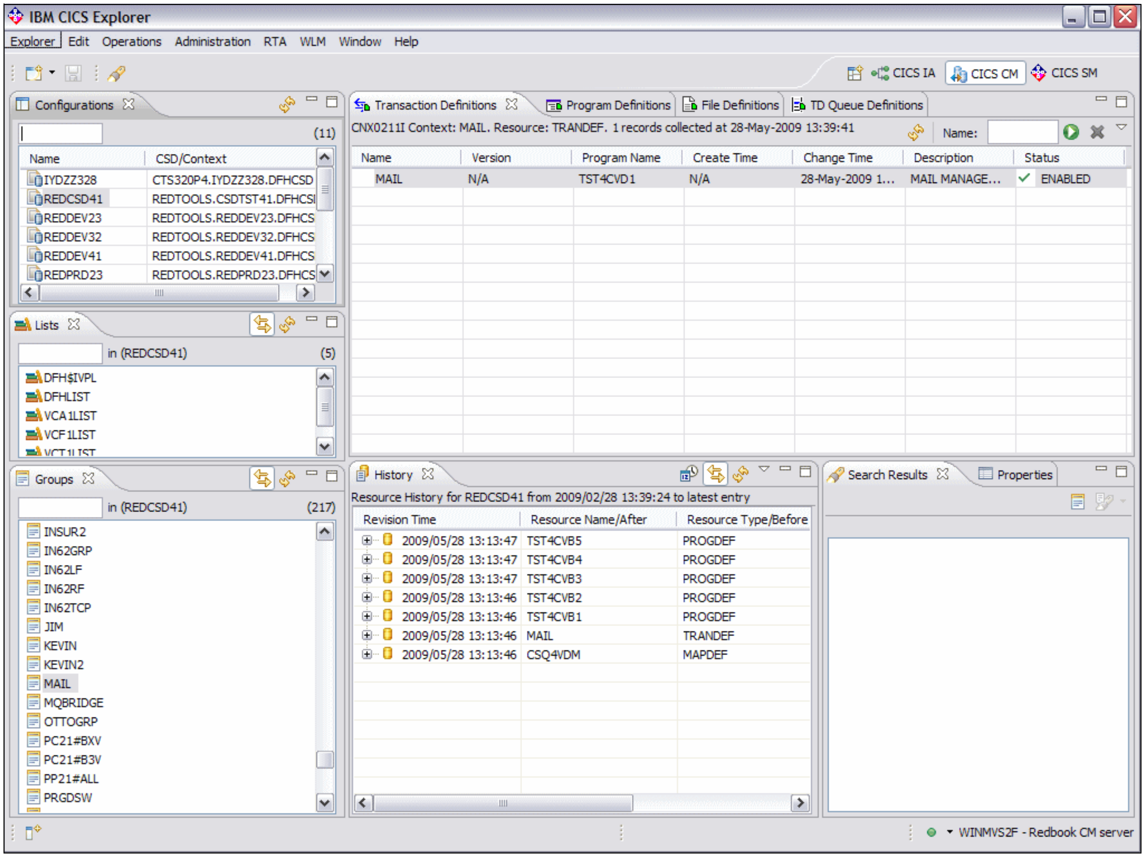


Figure 7-17 CICS Configuration Manager perspective

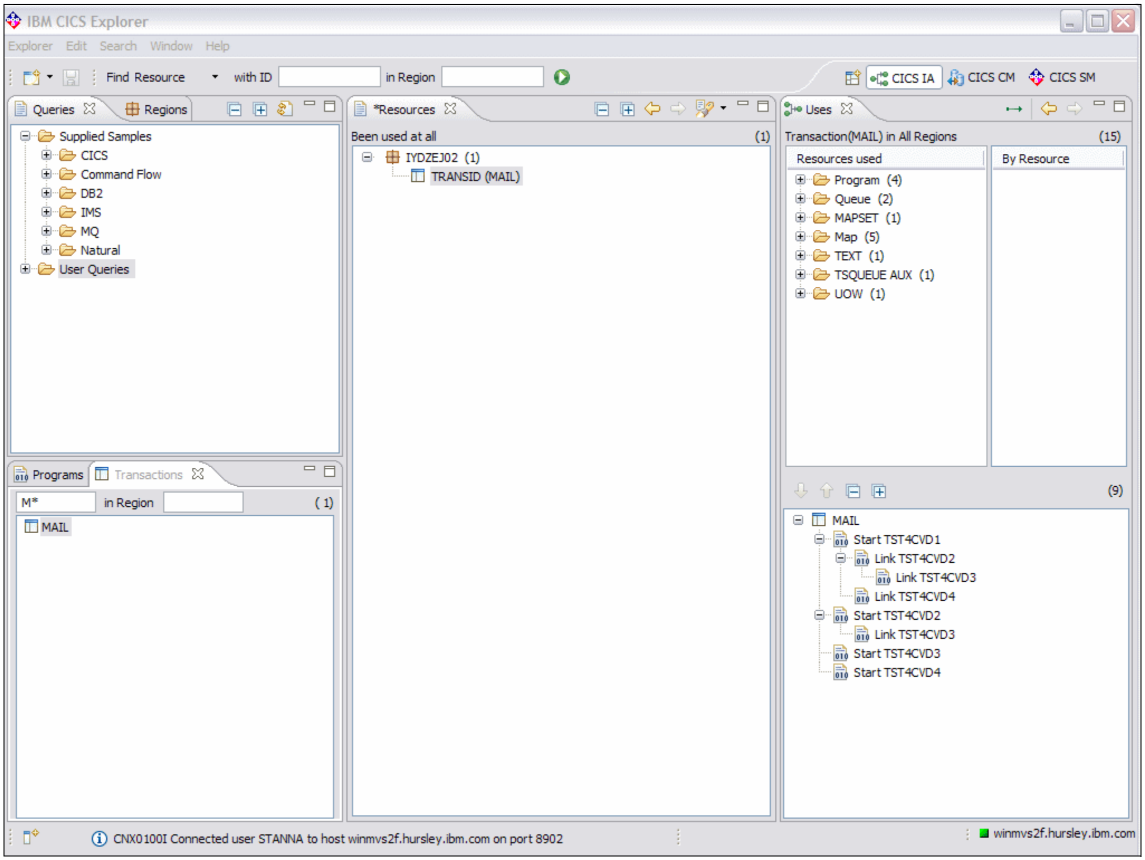


Figure 7-18 CICS Interdependency Analyzer perspective

The CICS Explorer is based on the Eclipse rich client platform (RCP). The Eclipse RCP is a cross-platform framework for building and deploying PC desktop applications. One of the features of Eclipse that makes it a good choice of technology for CICS Explorer is its plug-in architecture that allows you to add components that have the ability to extend and complement each other.

Today, many plug-ins exist that can be integrated into Eclipse environments, such as workflow, messaging clients, or development tools. One of these products is IBM Rational Developer for System z. Given the open nature of the Eclipse framework, clients and IBM Business Partners can also develop plug-ins for the CICS Explorer using the CICS Explorer software development kit (SDK).

With the emergence of CICS Explorer, the specialized skill requirements for supporting CICS environments are no longer an inhibitor to new application deployments.

7.7 Perspectives

CICS Explorer provides two perspectives for interacting with CICS:

- Resource

  The Resource perspective allows you to manipulate event bindings and bundle projects.

- CICS SM

  The CICS SM perspective allows you to view and manipulate CICS resources. This perspective is the focus of this chapter.

7.8 Connections

In this section, we discuss connections.

7.8.1 Connection manager

In CICS Explorer, the connection manager shows the status of any connections. The connection manager resides in the lower right corner of the CICS Explorer window, as shown in Figure 7-19.

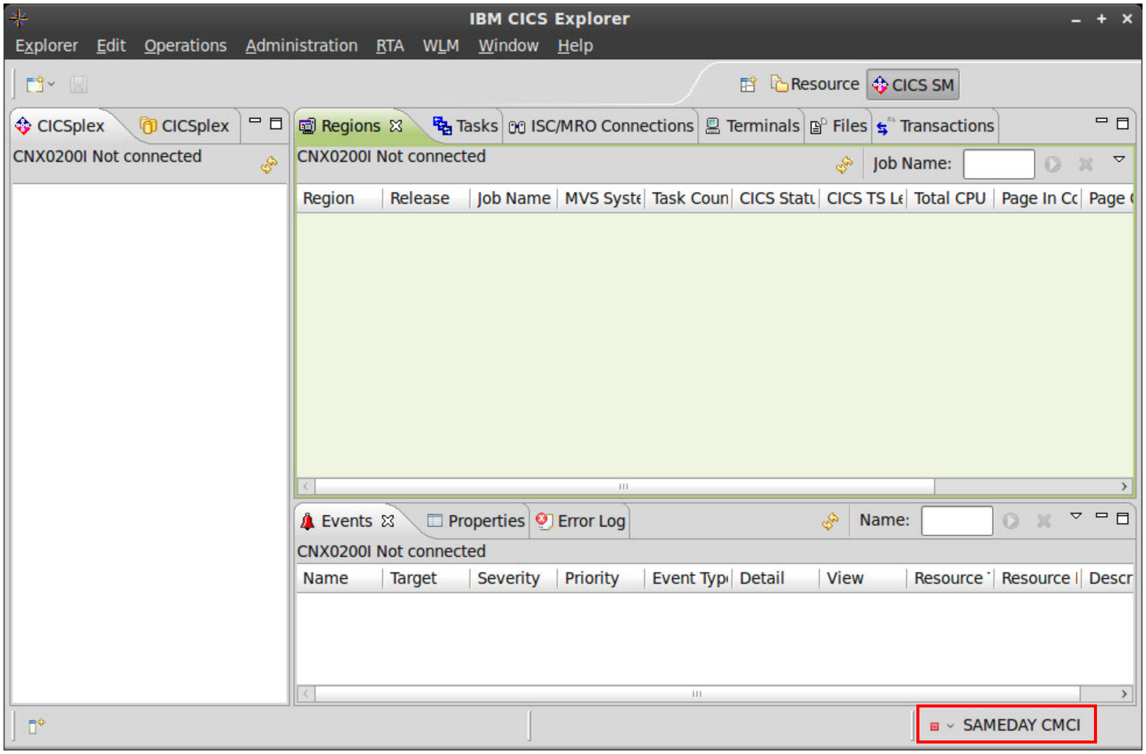


Figure 7-19 Connection manager location

When you configure CICS Explorer with connections, clicking the arrow on the connection status trim widget shows a list of available connections. Selecting one of these connections causes the CICS Explorer to attempt to establish that connection using the saved details. The connection manager icon represents the state of the named connection. Table 7-1 shows the connection manager status icons.

Table 7-1 Connection manager status icons

Connection status icon	Connection state
	Disconnected
	Connecting
	Connection established
	Secure connection established
	Connection error

When you click the red square, the system attempts to connect to the current named connection. While connecting, the icon changes to a yellow diamond. A green circle represents a successfully established connection. When a connection is established, clicking the green circle causes CICS Explorer to disconnect.

7.8.2 Connections Preferences page

You use the Connections Preferences page to manage the connections in CICS Explorer.

Follow these steps to access the Connections Preference page:

1. Click the text on the connection status trim widget, or use the **Window** → **Preferences** menus to open the preferences dialog.
2. From the tree on the left side, select **Connections**, as shown in Figure 7-20.

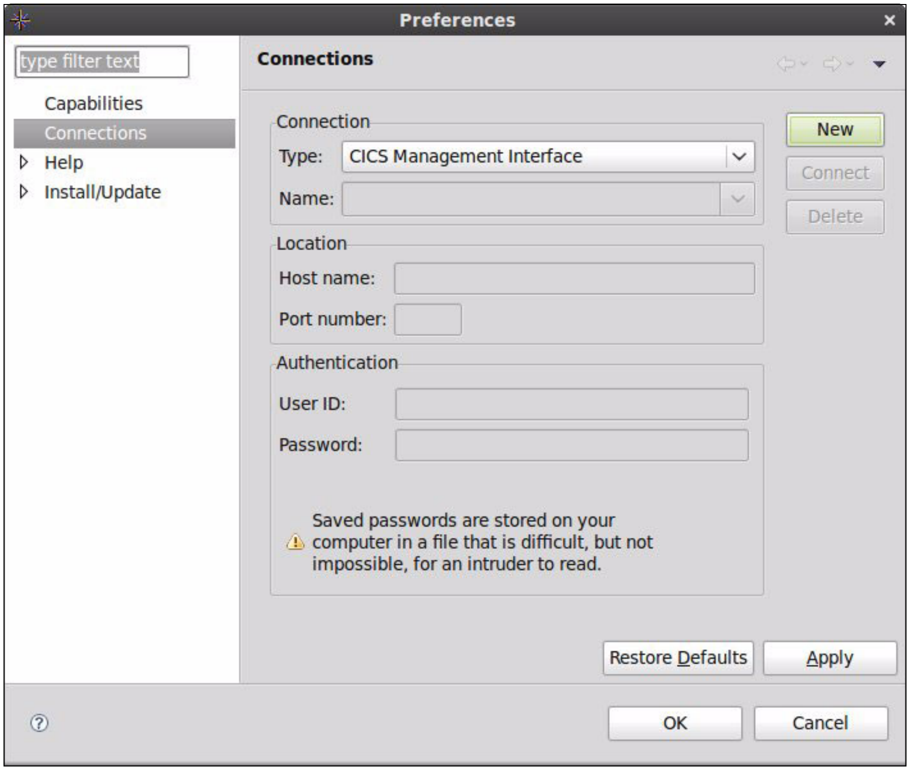


Figure 7-20 Connections Preferences page

Follow these steps to enter the details for a new connection:

1. Select the type of connection that you want to define, and click **New**.
2. After you specify your connection details, click **Apply** to save them.

Connections are categorized in the Preferences page based on their connection type. To view the available connections for a specific connection type, in the Type combo, select the type of connection that you want to examine. The Name combo is then populated with a list of connections of that type from which you can select.

The Type, Name, Host name, Port number, and User ID fields are mandatory. You can optionally specify passwords for authenticated connections. If a password is not specified, CICS Explorer prompts you for a password when establishing the connection.

7.8.3 CICS Explorer connectivity

In CICS Explorer, we click **Window** → **Preferences**, and select the connection settings field, as shown in Figure 7-21.

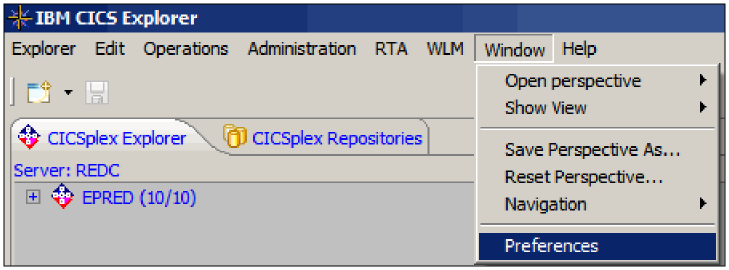


Figure 7-21 CICS Explorer Preferences

We select connection type **CICS Management Interface (CMCI)**, as shown in Figure 7-22.

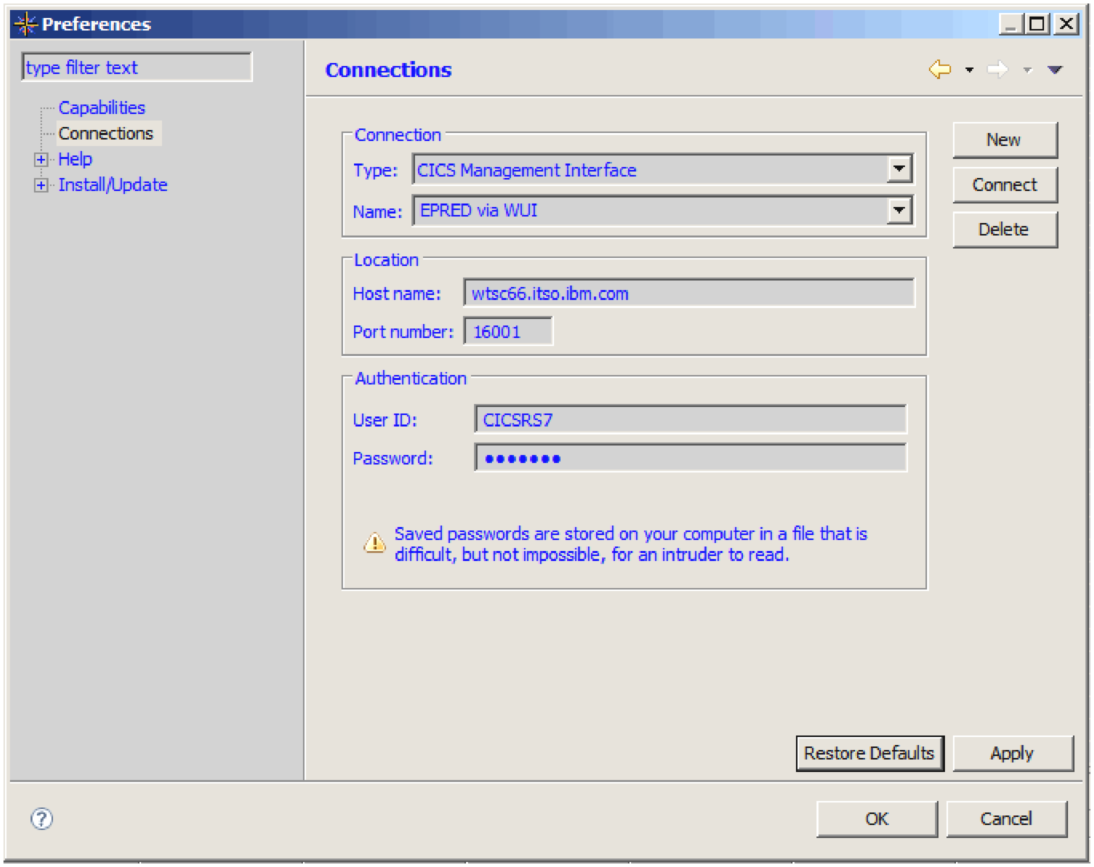


Figure 7-22 CICS Explorer connection options

> **Update capability:** We use the CICS Management Interface connection type to enable update capability. The default connection type, CICSPlex SM Data Interface, allows read-only capability.

Our web user interface (WUI) server (Host name: `wtsc66.itso.ibm.com`) has port = `16001`. The CMCIPORT system initialization parameter for our WUI is set as `CMCIPORT =16001`.

For further details, in the CICS Transaction Server for z/OS Version 4 Information Center, read *Setting up the CICS management client interface*:

https://publib.boulder.ibm.com/infocenter/cicsts/v4r1/index.jsp?topic=/com.ibm.cics.ts.clientapi.doc/topics/clientapi_setup.html

7.9 Views

In this section, we look at the two types of views: tabular and tree.

The context limits what data is displayed in most views:

- You can manipulate the context of a view by the CICSplex Explorer view or the CICSplex Repositories view.
- You can use the CICSplex Explorer view to set the context to a CICSplex or CICS region.
- You can use the CICSplex Repositories view to set the context to a resource group or system group.
- You can also use the Resource Group Definitions view to set the context to a resource group.

CICS Object Table views use the last appropriate context that the user selected when populating. For example, only CICS definition views repopulate using a resource group because their context is selected by the user.

7.9.1 The CICSplex Explorer view

When connected to CICSPlex SM, the CICSPlex Explorer view (Figure 7-23) shows a list of CICSPlexes. When using CMCI to connect to a single CICS region, the CICSplex Explorer view shows the region to which you are connected.

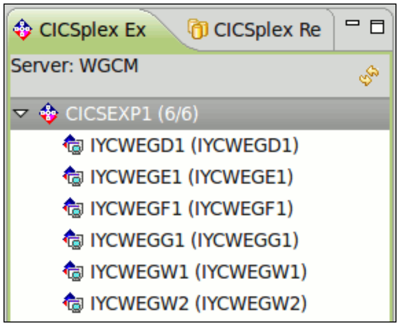


Figure 7-23 The CICSplex Explorer view

Expanding a CICSplex shows the managed regions within that CICSplex. If the icon is colored, the managed region is active. If the icon is monochrome, the region is inactive.

Selecting a CICSplex or a managed region sets the context accordingly.

7.9.2 The CICSplex Repositories view

The CICSplex Repositories view, Figure 7-24, shows the available resource groups, the available system groups, and their members.

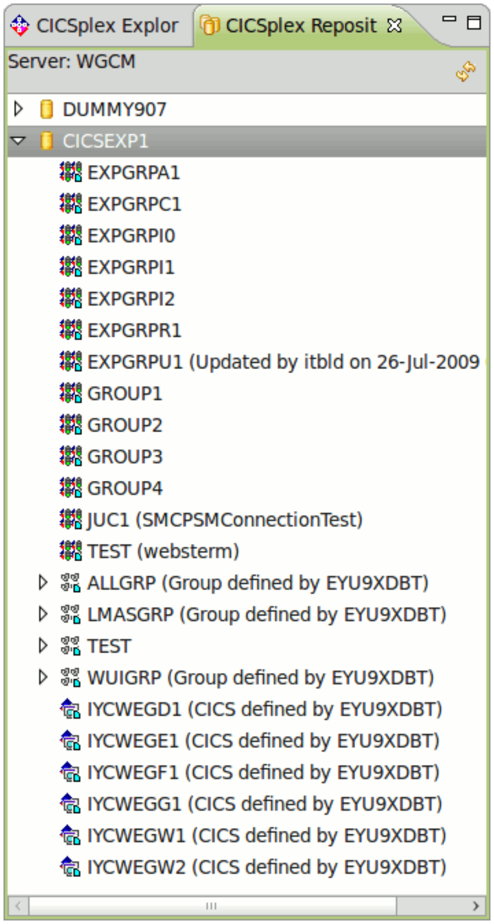
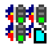


Figure 7-24 CICSplex Repositories view

Table 7-2 shows the icons for the CICSplex Repositories view.

Table 7-2 CICSplex Repositories view icons

Icon	CICS object
	System definition
	System group
	Resource group

Selecting any of these resources in the CICSplex Repositories view sets the context of any open CICS Object Table view where appropriate. You can expand system groups to view their members, which might be either nested system groups or system definitions.

7.9.3 CICS Object Table views

CICS Object Table views display CICS resources of a specific type, from the current context. You cannot open Resources views from the Administration, Operations, Workload Manager, and real-time analysis menus.

7.10 The value of the new tools

In the past, individuals used the specific skill requirements for mainframe application development as an argument against the deployment of new applications on the mainframe. Yet, the skills that are required for developing mainframe applications with the latest integrated development environments (IDEs) are now extremely similar to other environments.

For CICS application developers, Rational Developer provides a single, integrated IDE for all supported languages and architectural styles whether you are creating new CICS applications from scratch or are wrapping, refactoring, or otherwise reusing your CICS application assets.

Figure 7-25 shows a pictorial view of how Rational Developer can assist in the development of CICS COBOL programs.

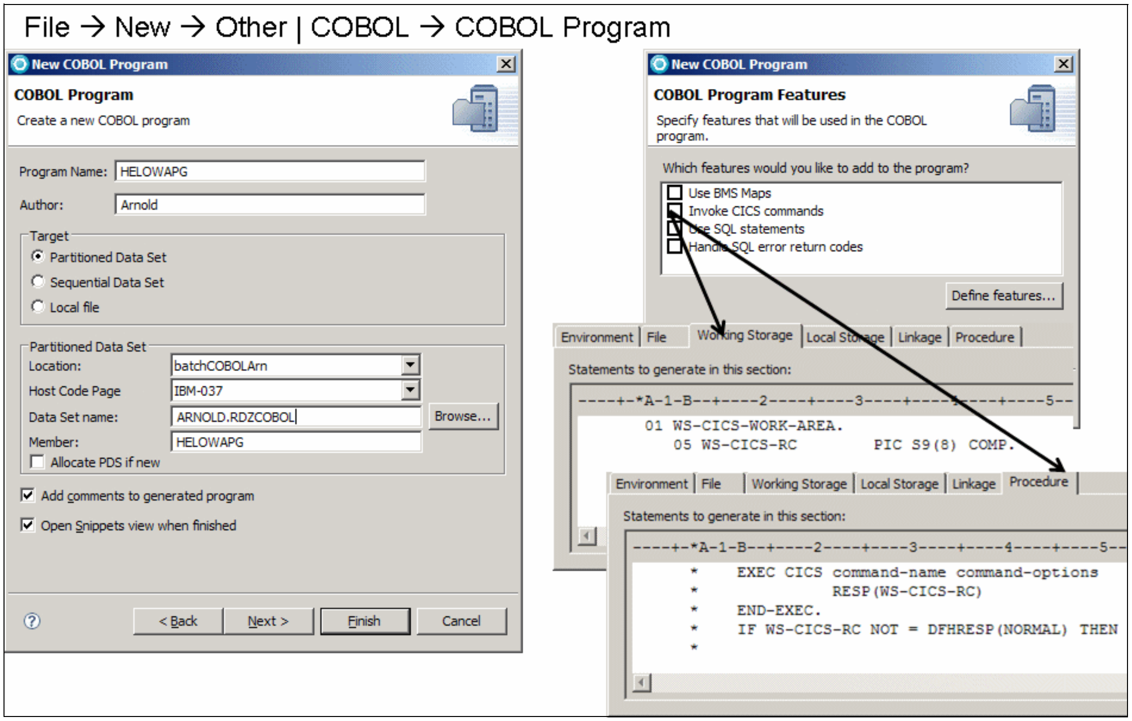


Figure 7-25 Using Rational Developer to develop CICS COBOL programs

Rational Developer supports end-to-end, model-based development, runtime testing, and rapid deployment of mixed-language applications. You can create application components in various programming languages and deploy them to a range of runtime environments, including CICS and WebSphere Application Server.

Using Rational Developer, you can develop core business applications more easily in one of the supported languages: COBOL, PL/I, C/C++, and Java. You can then deploy these applications as service components and integrate them using standard interfaces, for example, Web Services.

The latest version of Rational Developer includes support for CICS Explorer. All the systems management tools previously had their own separate user interfaces, CICS 3270 transactions, web browser, Time Sharing Option (TSO) ISPF panels, or batch jobs. When you consider the enormous number of third-party, vendor, and client in-house-developed CICS systems management tools, each with its own user interface and implementation, you can see why CICS system administrators and programmers tend to regard with suspicion yet another tool with yet another user interface to learn.

CICS Explorer signals an end to this confusion. It allows all the tools and applications that a user needs to perform a business task to present the same structure and user interface. Tools that are built by IBM, vendors, and clients extend and enhance CICS Explorer in a consistent way, adding value to CICS management without the need to reinvent or redefine CICS Explorer's user interface.

8

Reference application

In this section, we describe the reference application that we created for the use cases. The application source code is available as softcopy on the Internet from the IBM Redbooks web server. Point your web browser at this location:

`ftp://www.redbooks.ibm.com/redbooks/SG247868`

The reference application is a scaled-down facsimile of a production application, and it includes the following components:

- Job control language (JCL)
- COBOL programs and copybooks
- IBM WebSphere MQ for z/OS
- DB2 tables
- CICS online programs
- Queued sequential access method (QSAM) and Virtual Storage Access Method (VSAM) files

Most of the application is batch (JCL). We based the application design on our combined experiences. The domain is a health insurance model.

> **Important:** This application is not actually a production application. We use it in this book only for illustrative purposes.

8.1 Nightly batch cycle system flow chart

In Figure 8-1, you can see that the nightly batch cycle consists of processing that creates generation data group (GDG) data sets consisting of patient treatment data collection files and inpatient hospital room charges. Both inpatients and outpatients receive treatment, but only inpatients can have room charges.

The concept of *data collection* refers to transactional data that is collected in real time or online during the day by IBM Information Management System Transaction Manager (IMS TM) or CICS. These data collection files represent non-normalized (raw) data records that are applied during a nightly database or master file update process.

Typically, the processing of data collection applications consists of these actions:

- Edits: Both trivial and complex edits that scrub the transaction data
- Enhancements to the files: Perhaps applying additional table-based information, or extending the initial record values by appending other related file information
- Sort/merge: Collating the records according to the system's business rules
- Updates to the master files, databases, and tables: By applying data collection record information
- Reporting: Writing out online (originally paper) reports for business users and back-end office requirements (accounting trails and so forth)

  Note that these steps often require additional sorting on the report collating requirements, for example, in our case, ward, room, patient, and so on

Historically, shops ran the batch cycle for these types of applications overnight. The shops used JCL streams that were scheduled through IBM Tivoli® or another batch application scheduling package that launched jobs in a sequence based on information that needed to be processed before other information was processed, file dependencies, and so forth. Our nightly batch is simply one step in such a process.

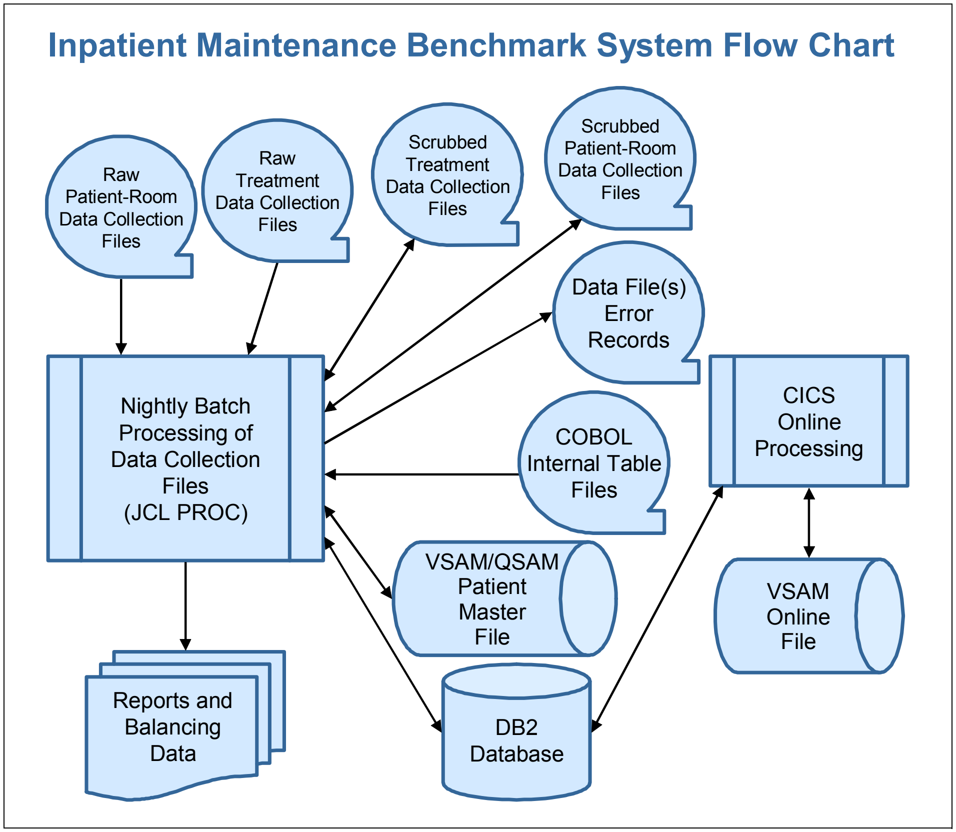


Figure 8-1 System flow chart

8.2 Data sets used in the application

In the next few sections, we describe the various files and data sets that are used in the reference application. IBM Rational Asset Analyzer produced all the documentation shown here. Producing application documentation directly from the production source code seems to us the only defensible means of doing this kind of work if you have access to the tools of the optimized life cycle. Not only is it the most accurate documentation that you can produce, it literally takes around one second per snapshot to create, making it the best project documentation experience we can recall.

Figure 8-2 shows a partial list of the 45 QSAM (GDGs) and VSAM key-sequenced data sets (KSDS) that are used by the application.

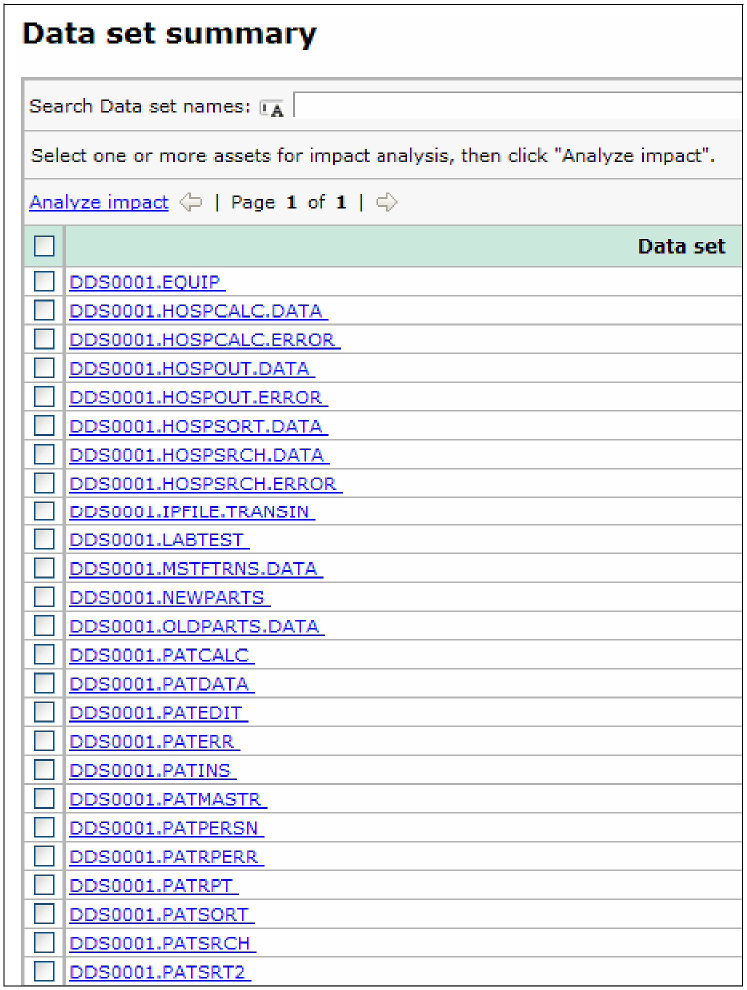


Figure 8-2 List of QSAM and VSAM data sets produced by Rational Asset Analyzer

8.2.1 DB2 tables

Figure 8-3 and Figure 8-4 show the DB2 tables in the reference application. Most of the tables' usage model was as information lookup, rather than using DB2 to store transactional data. Many older production batch jobs still execute this way, using QSAM and VSAM files (or IMS Fast Path databases) for data collection and then updating master tables, files, and databases in batch.

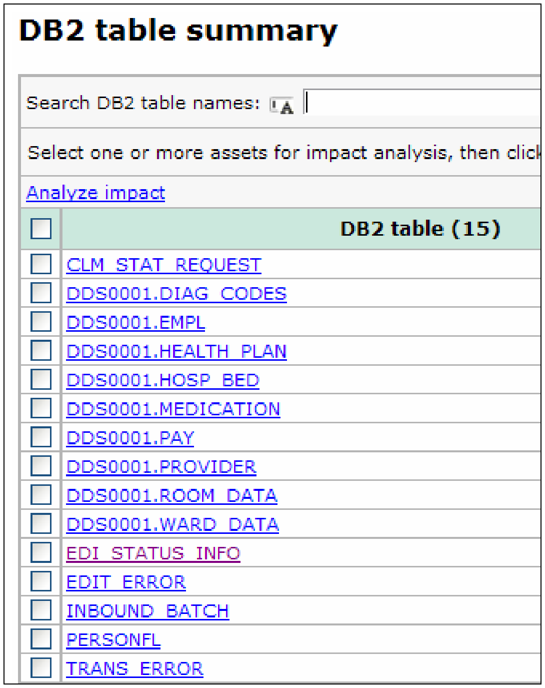


Figure 8-3 List of DB2 tables

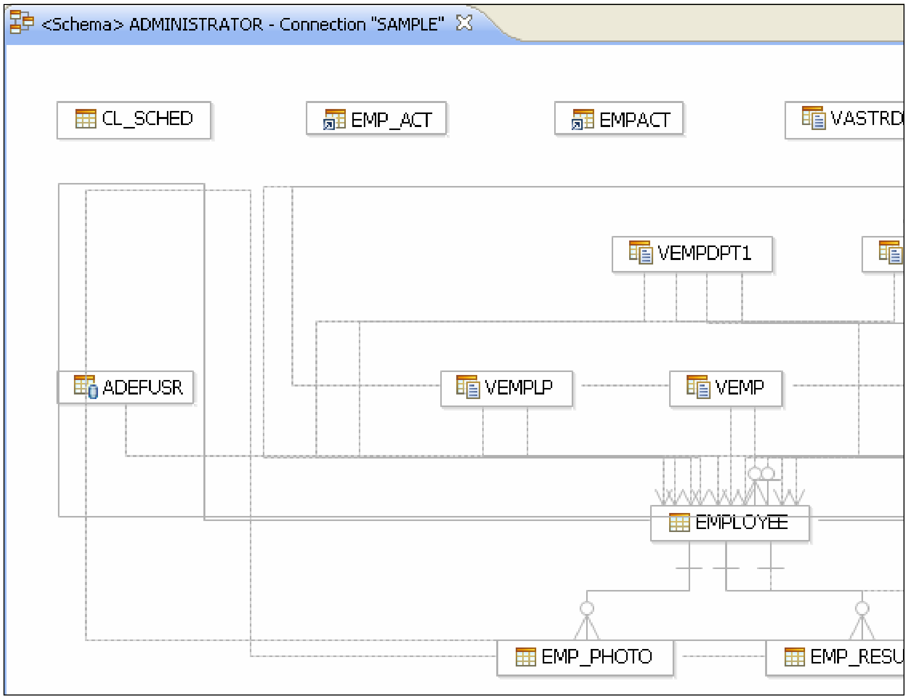


Figure 8-4 Partial DB2 entity-relationship model for the tables in the reference application

8.2.2 JCL for the batch cycle

The heart of the reference application is the batch cycle, and JCL runs z/OS batch. Figure 8-5 on page 413, Figure 8-6 on page 414, Figure 8-7 on page 415, and Figure 8-8 on page 416, all of which were produced by Rational Asset Analyzer, show several of the batch steps. You can see by the screen captures that this batch is extremely small by production standards. We have chosen to focus on the development process and to keep the level of details to a manageable scale.

Figure 8-5, Figure 8-6, and Figure 8-7 show several of the steps. We do not show the complete batch job processing diagram of the inter-step file and table dependencies due to the limited page size of this book.

However, the Rational Asset Analyzer Batch Job Diagram tool is exceptionally easy to use for the purposes of understanding job flows and data set usage.

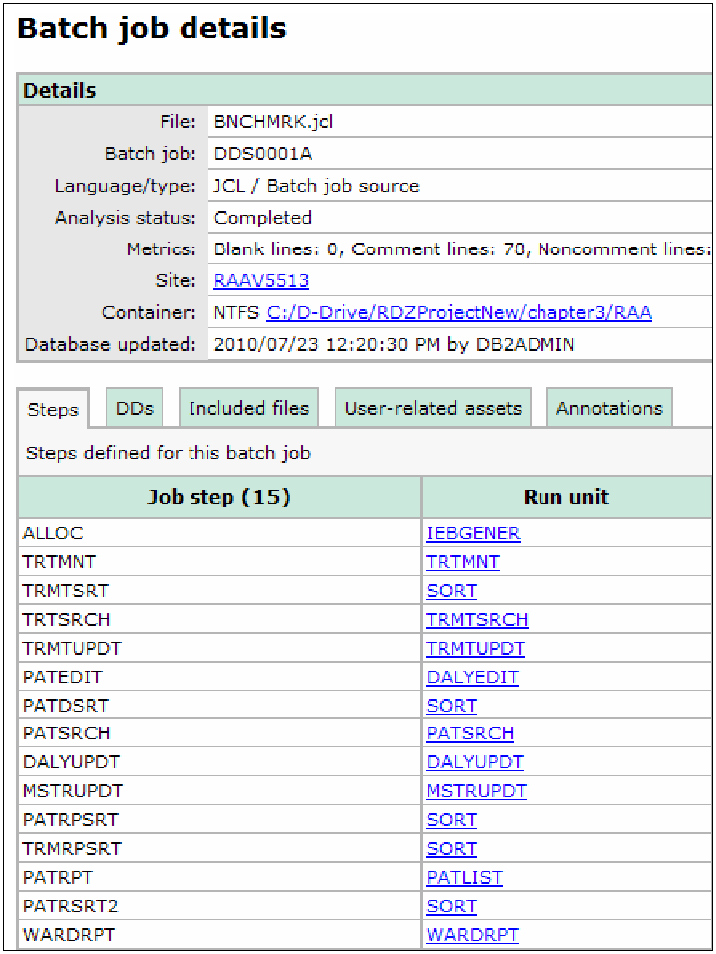


Figure 8-5 Batch job stream: Steps and run units

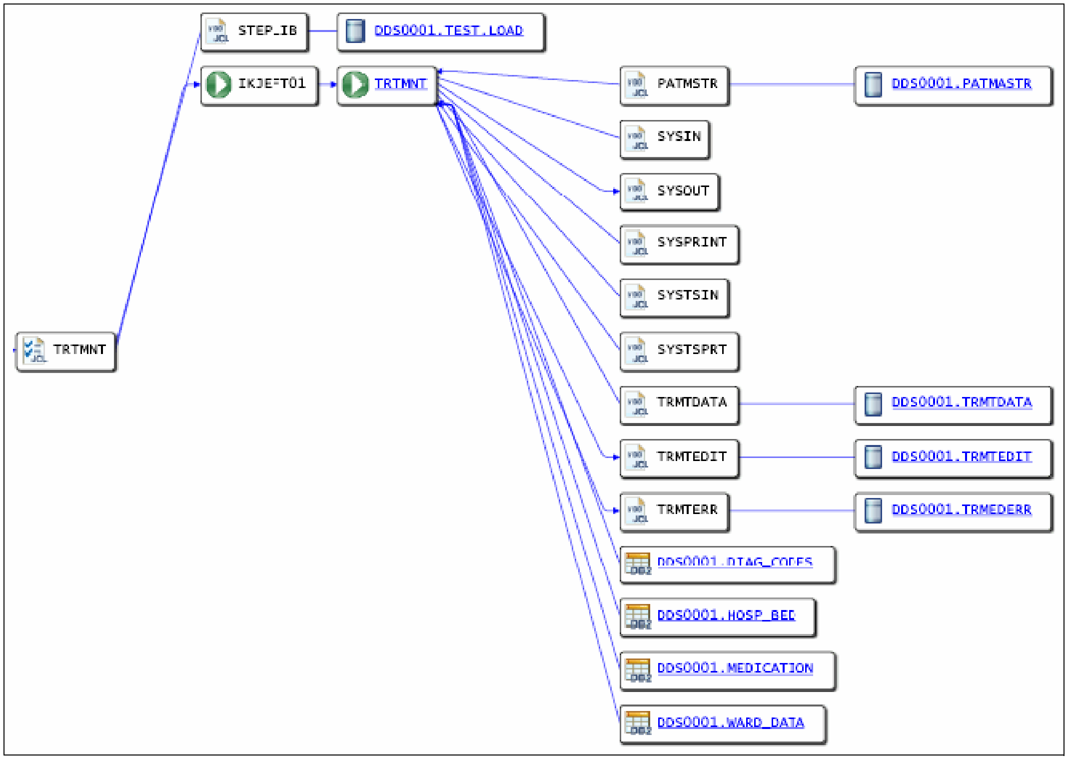


Figure 8-6 TRTMNT job step (arrows depict I/O direction) using DB2 tables and QSAM and VSAM files

From Figure 8-6, you can see that the TRTMNT job step launches the Time Sharing Option (TSO) command processor (IKJEFT01) to run this DB2 batch module from the DDS0001.TEST.LOAD load library.

The program TRTMNT reads YDDS0001.PATMSTR, TRMTDATA, SYSIN, SYSTSIN, SYSTSPRT, and the four DB2 tables and writes DDS0001.TRMTEDIT and TRMTERR.

Sometimes, the `sysin` and `sysout` (DD *) files are not sufficiently important to include in the production documentation. In that case, Rational Asset Analyzer can hide those data sets from the job step diagram (Figure 8-7).

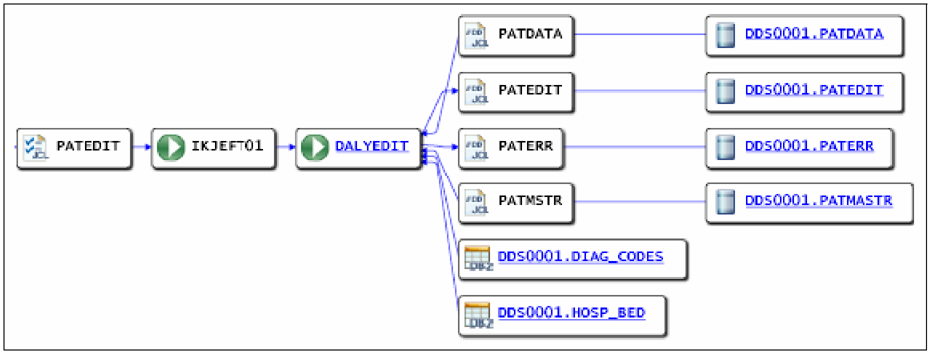
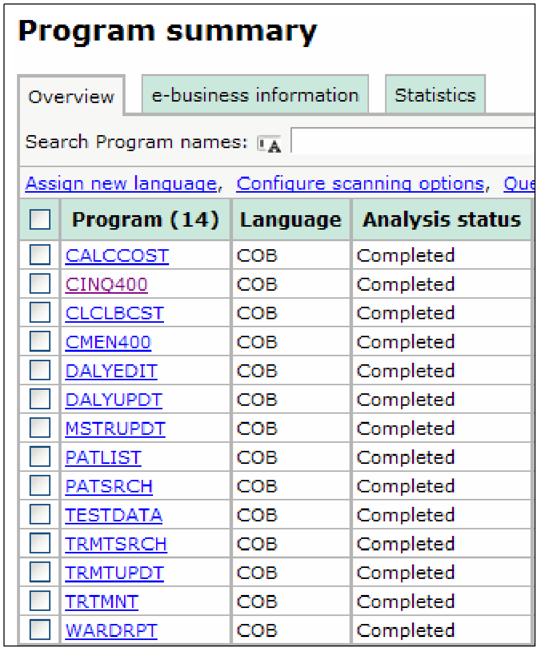


*Figure 8-7 PATEDIT job step (Rational Asset Analyzer has removed STEPLIB and DD * I/O files)*

8.2.3 COBOL programs

Figure 8-8 lists several of the COBOL programs that are used in the job stream. There are also several called Assembler modules. John Milne, one of the authors, created calls to the WebSphere MQ messaging system from the COBOL program (MSTRUPDT). We have not included the WebSphere processing in this documentation; however, you can read the WebSphere processing details in Chapter 9, “WebSphere MQ and the optimized life cycle” on page 423.



	Program (14)	Language	Analysis status
	CALCCOST	COB	Completed
	CINQ400	COB	Completed
	CLCLBCST	COB	Completed
	CMEN400	COB	Completed
	DALYEDIT	COB	Completed
	DALYUPDT	COB	Completed
	MSTRUPDT	COB	Completed
	PATLIST	COB	Completed
	PATSRCH	COB	Completed
	TESTDATA	COB	Completed
	TRMTSRCH	COB	Completed
	TRMTUPDT	COB	Completed
	TRTMNT	COB	Completed
	WARDRPT	COB	Completed

Figure 8-8 Fourteen of the COBOL programs that are used in the batch stream

Figure 8-9 and Figure 8-10 show the DALYEDIT program. From this starting point, you can explore high-level information about the program:

- Number of includes (Figure 8-9)
- DB2 tables, columns, and stored procedures that are referenced
- Batch jobs that invoke the program
- Data set usage
- Calls to and from the program (Figure 8-10), including details, such as the call sequence and the parameters that are passed

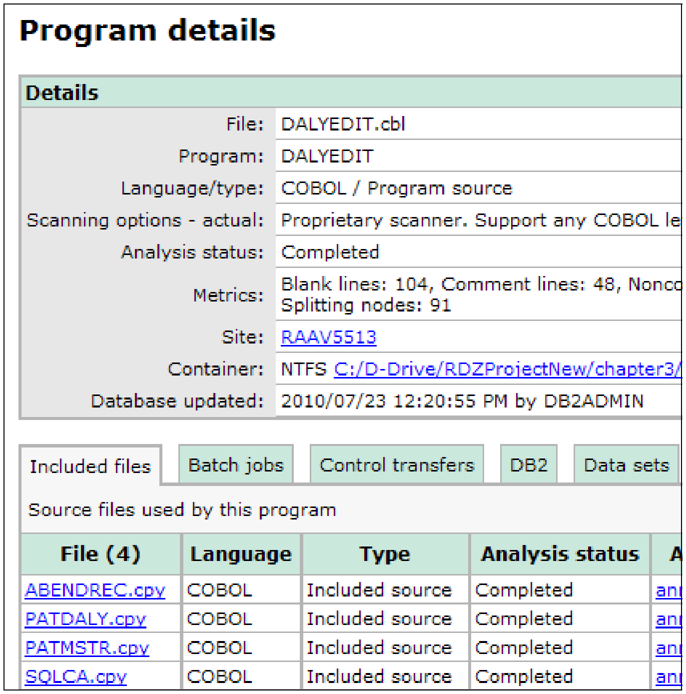


Figure 8-9 DALYEDIT COBOL program listing copy files

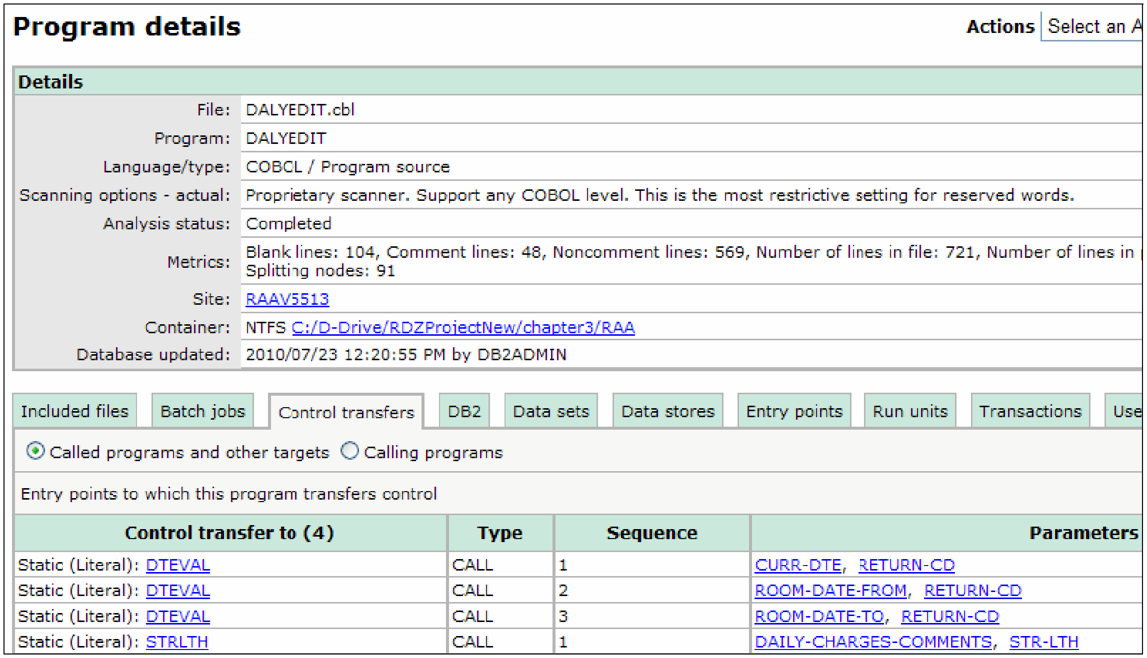


Figure 8-10 Control transfer within DALYEDIT: Note static transfer (versus dynamic (variable assignment)), the sequence of calls, and on the right, the call parameter variables

Rational Asset Analyzer also shows more graphical views of programs. See Chapter 5, “Rational Asset Analyzer” on page 199 for details about Rational Asset Analyzer’s capabilities.

Figure 8-11 shows a run unit diagram that depicts how programs are launched from the batch job's JCL, I/O to files, tables, and databases, as well as calls to subprograms (if any exist).

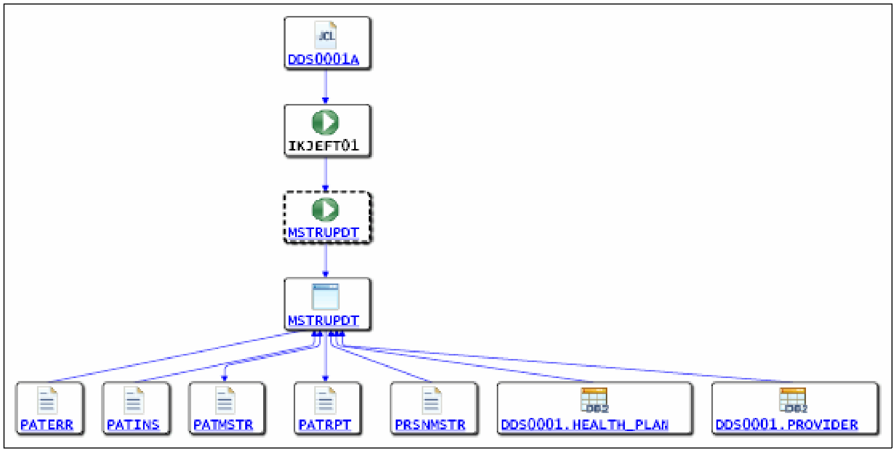


Figure 8-11 MSTRUPDT run unit diagram showing launch sequence and file and database I/O

8.2.4 Batch reports

The batch programs in this application (there are two) roll up two types of data:

- Patients:
  - Detailed information
  - Room charges
  - Treatment charges
- Ward utilization: A breakdown of the hospital's wards, rooms, and beds.

Both of these programs are control-break style reports, where data comes in sorted by major, mid-level, and minor-level values for proper matching and processing logic.

Figure 8-12 shows the output of one of the programs.

```
10-01-2009                         Patient Detailed Treatments and Charges List                         Page #

ID      Name           Phone  Type       Ward   Room   Bed   Admit Date  Diagnosis  Physician       Daily Charge  Type   Provider
00001   Smith, George  4324   Inpatient  1234   234    03    12-20-2009  DIAG1      Smith, Elmore   $$,$99.99     HMO    CIGNA

  Equipment charges:
    Equipment: xxxxx    Category: xxxxxx   Cost: $,$99.99  Short Description: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    Equipment: xxxxx    Category: xxxxxx   Cost: $,$99.99  Short Description: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    Equipment: xxxxx    Category: xxxxxx   Cost: $,$99.99  Short Description: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

  Treatments: Medication: xxxxx   Medication Cost: $,$$$,$$$.99   %Covered:  999   Comments: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    Lab-Test: xxxxx     Category: xxxxxx   Cost: $,$99.99  Venipuncture:  $,$$9.99   Description: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    Lab-Test: xxxxx     Category: xxxxxx   Cost: $,$99.99  Venipuncture:  $,$$9.99   Description: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    Lab-Test: xxxxx     Category: xxxxxx   Cost: $,$99.99  Venipuncture:  $,$$9.99   Description: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

  Treatments: Medication: xxxxx   Medication Cost: $,$$$,$$$.99   %Covered:  999   Comments: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    Lab-Test: xxxxx     Category: xxxxxx   Cost: $,$99.99  Venipuncture:  $,$$9.99   Description: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    Lab-Test: xxxxx     Category: xxxxxx   Cost: $,$99.99  Venipuncture:  $,$$9.99   Description: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    Lab-Test: xxxxx     Category: xxxxxx   Cost: $,$99.99  Venipuncture:  $,$$9.99   Description: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                 Patient Summary – Total length of stay: 999   Total costs:  $,$$$,$99.99
```

Figure 8-12 Patient Detailed Treatments and Charges List report

8.3 CICS online programs

There are two simple CICS programs in this application: Patient Menu and Patient Inquiry:

- The menu program presents a standard green screen using basic mapping support (BMS), as shown in Figure 8-13.
- The inquiry program allows an operator to enter a unique Patient-ID, which is then used to look up patient information that exists in various VSAM files (Figure 8-14).

The processing logic in these programs is extremely simple (COBOL, CICS, and VSAM). Our purpose in this book is to show how the tools in the optimized life cycle can extend to various areas of application source when maintenance and support tasks are required.

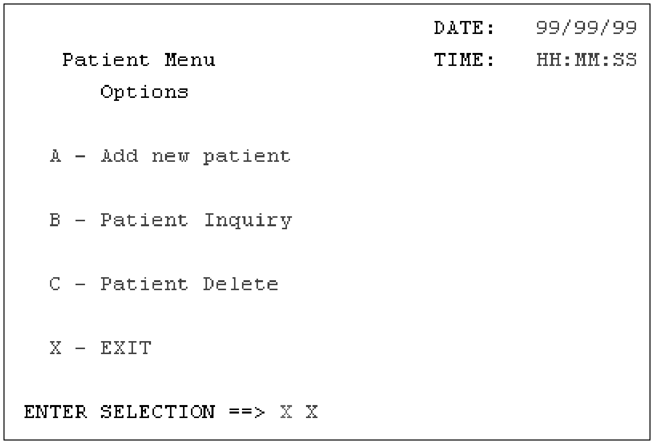


Figure 8-13 CICS 3270 menu

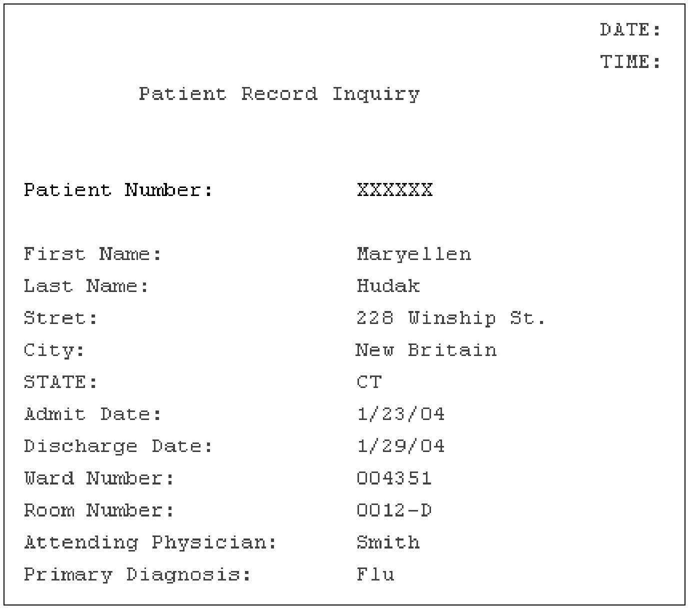


Figure 8-14 Patient Record Inquiry 3270 window

8.4 Summary

In this chapter, we have described the reference application that is used in this book. To obtain a copy of any of the assets in the reference application, see Appendix B, “Additional material” on page 599.

9

WebSphere MQ and the optimized life cycle

In this chapter, we provide background information about the IBM messaging middleware product, WebSphere MQ. Understanding WebSphere MQ is not necessary to use the elements of the optimized life cycle, such as IBM Rational Developer for System z, IBM Rational Asset Analyzer, or IBM Rational Team Concert for System z. The information in this chapter is designed to give those individuals who need to work with WebSphere MQ an insight into the development and maintenance of WebSphere MQ-enabled applications using the tools and processes that are associated with the optimized life cycle.

If you are already familiar with WebSphere MQ, you can skip this chapter. If you have been around, but not close-up to WebSphere MQ, this chapter can provide a starting point for a more in-depth understanding and appreciation of the power of this important part of many production sites today.

This chapter contains the following topics:

- WebSphere MQ overview
- WebSphere MQ terms and concepts
- WebSphere MQ and Rational Developer
- WebSphere MQ debug example
- Comparisons between developing with Rational Developer and ISPF

9.1 WebSphere MQ overview

WebSphere MQ is a product that enables applications running on a myriad of platforms to exchange messages through a common set of Application Programming Interfaces (APIs) that are known in the WebSphere MQ documentation as the *Message Queue Interface* (MQI). WebSphere MQ (IBM MQSeries®) began its evolution with Version 1.1, which was made generally available at the end of 1993 through to Version 7.0, which was delivered in the middle of 2008. The product provides a consistent, flexible, and reliable mechanism to deliver messages between applications (Figure 9-1).

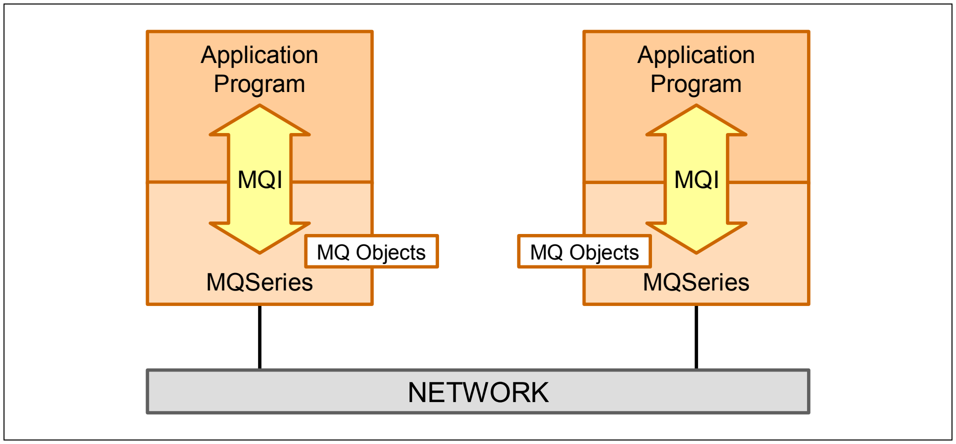


Figure 9-1 MQSeries at run time

9.2 WebSphere MQ terms and concepts

We explain many of the common terms that you encounter in WebSphere MQ documentation:

- Message: A *message* is application data that one application program needs to send to another application program. The data can be a request for information, it can be a reply to a request, or it can be a one-way notification of an event. There are two components to a WebSphere MQ message:
  - *Message data* is the information that the application needs to send to another application. The application defines the contents and structure of the message data. WebSphere MQ does not need to access the contents to ensure that the message is delivered as the application requires.

  - The *message metadata* is known as the MQSeries Message Descriptor (MQMD), which contains information about the message. The application determines the settings within the MQMD. The application data is prefixed by the MQMD, and it accompanies the message on its journey from application to application through WebSphere MQ.

- Queue manager: The *queue managers* are the nodes with a WebSphere MQ network. The queue manager is responsible for housing messages either as an endpoint or on an interim basis before they are subsequently passed on to another queue manager.
- One important message property that is set in the MQMD is whether the message is *persistent* or *non-persistent*. When a persistent message is put to a queue, the queue manager externalizes the message to a permanent data storage device before it flags the application that the put was successful. This method ensures that persistent messages are not retained when a queue manager stops and restarts. This method does not happen for a non-persistent message, and non-persistent messages are not retained when a queue manager stops and restarts.
- Clients: WebSphere MQ client software allows applications running on a workstation to connect to a queue manager and work with WebSphere MQ messages.
- Channels: WebSphere MQ objects connect queue managers with each other, allowing the delivery of messages from one queue manager to another, which typically means from one server or platform to another server or platform. Channels are unidirectional. For bidirectional communications, you must define a pair of channels. A special type of channel, which is called an *MQI channel link*, exists between an MQ client and a queue manager.
- Processes: WebSphere MQ objects define an application to WebSphere MQ. You use a *process* when you are required to triggering actions on the arrival of a message on a queue. For example, a process might specify the CICS transaction that is to be started.
- Triggers: You can configure WebSphere to start an application when a message arrives on a queue. You can configure WebSphere to *trigger* an action when the first message arrives (queue depth moves from zero to one), every time that a message arrives, or when the queue depth reaches a certain value.
- Message Queue Interface (MQI): An *MQI* is the programming interface that is provided by WebSphere MQ queue managers. The programming interface allows application programs to access message queuing services. There are two groups of MQIs. The first group provides simple functionality:
  - Connect to a queue manager: MQCONN and MQCONNX (extended)
  - Disconnect from a queue manager: MQDISC

- Open a queue, which is similar to opening a file: MQOPEN
- Close a queue, which is similar to closing a file: MQCLOSE
- Put a message to a queue, which is similar to writing a record to a file: MQPUT
- Get a message from a queue, which is similar to reading a record from a file: MQGET
- Open a queue, put a message to the queue, and close the queue, all in one command: MQPUT1

- The second group provides commands to view and manipulate WebSphere MQ objects (queues, processes, channels), as well as to control unit of work activity:
  - View the properties of an object: MQINQ
  - Change the properties of an object: MQSET
  - Begin a unit of work: MQBEGIN
  - Commit a unit of work: MQCMIT
  - Back out a unit of work: MQBACK
  - You can use MQBEGIN, MQCMIT, and MQBACK to ensure that actions with WebSphere MQ and a database management system (DBMS) are coordinated to either all work or all fail, that is, to have atomic, consistent, isolated, and durable properties.

9.3 WebSphere MQ and Rational Developer

Consider the following example to illustrate working with WebSphere MQ-enabled applications with Rational Developer. A hospital has a z/OS batch-based system that is used to maintain its patient database. The hospital now wants to extend the system to feed updated information about patients to a data warehouse that runs on another platform. The system must send the information as the updates occur in near real time and in a secure and reliable manner.

Currently, the COBOL program MSTRUPDT updates the patient master file using Virtual Storage Access Method (VSAM). The application sends a message to the data warehouse each time that a patient master file record is updated. It is not desirable to make the batch job dependent on a response from the data warehouse indicating that the data warehouse has received a message, but it is a requirement that no updates are lost. The hospital decides to use WebSphere MQ persistent messages to meet both of these requirements. The programmer adds calls to a new subprogram, MQPUTSUB, to the MSTRUPDT program.

9.3.1 Calling MQPUTSUB

Interfacing to WebSphere MQ in batch typically involves these actions:

- Connecting to the queue manager
- Opening the WebSphere MQ objects that we will work with
- Putting and getting messages
- Closing the WebSphere MQ objects
- Disconnecting from the queue manager

This coding pattern is similar to the coding pattern for working with files: open the file, perform many I/O operations to the file, and close the file. In this example, we put a single message to a queue. When a single message is put to a queue, we can use the WebSphere MQ MQPUT1 API, because it encapsulates MQOPEN, MQPUT, and MQCLOSE in a single command. We still have a requirement to connect (MQCONN) and disconnect (MQDISC), but we will connect and disconnect only one time for efficiency. After a connection is made to a queue manager, the queue manager supplies the application with a connection handle that uniquely identifies this connection between the application and the queue manager. The application must supply this connection handle to all subsequent MQ calls up to and including the corresponding MQDISC.

In this example, we confine all the WebSphere MQ API calls to the MQPUTSUB subprogram. Figure 9-2 shows the interface between MSTRUPDT and MQPUTSUB. MSTRUPDT calls MQPUTSUB one time in the initialization paragraph, 000-SETUP-RTN, to connect to the queue manager. The connection handle is returned to MSTRUPDT within the LN-COMM-BLOCK.

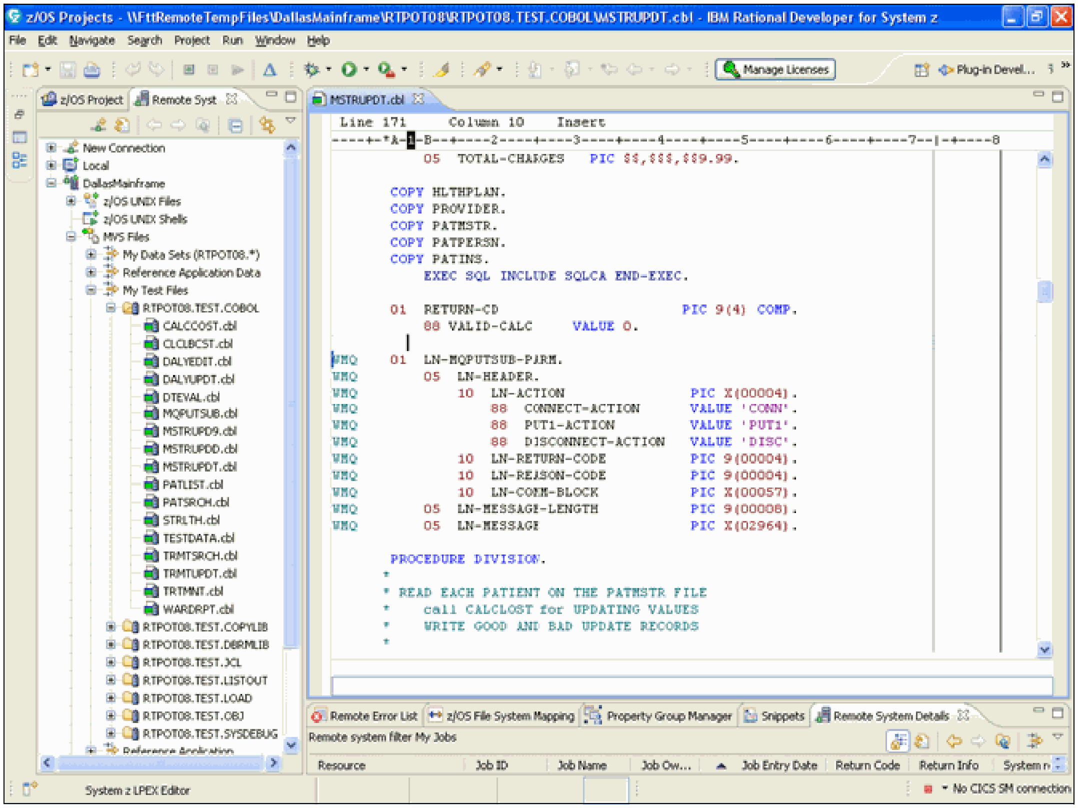


Figure 9-2 Rational Developer and a COBOL program with a WebSphere MQ copybook interface

In the major processing loop of MSTRUPDT, the application calls the subprogram MQPUTSUB to put the updated patient record to the queue that is bound for the data warehouse. Finally, the application calls the MQPUTSUB to disconnect from the queue manager in 900-CLOSE-FILES. Example 9-1 is an excerpt from MSTRUPDT that shows the call to MQPUTSUB to execute the MQPUT1 API.

Example 9-1 Call to MQPUTSUB to execute the MQPUT1 API

```
WMQ        MOVE PATIENT-MASTER-REC            TO LN-MESSAGE.
WMQ        SET PUT1-ACTION                    TO TRUE.
WMQ        CALL 'MQPUTSUB' USING LN-MQPUTSUB-PARM.
WMQ        IF LN-RETURN-CODE > 0
WMQ           DISPLAY "****************************"
WMQ           DISPLAY "* ERROR WITH WMQ PUT1       "
WMQ           DISPLAY "* MQPUTSUB:" LN-RETURN-CODE ":" LN-REASON-CODE
WMQ           DISPLAY "****************************"
WMQ           GO TO 1000-ERROR-RTN.
```

9.3.2 WebSphere MQ API calls

WebSphere MQ supplies COBOL copybooks to use when calling the WebSphere MQ APIs. These copybooks are located in the SCSQCOBC target library (SCSQMACS for assembler macros). When working with WebSphere MQ programs in Rational Developer, update the property group that is associated with the program to include the SCSQCOBC library in the copy libraries (Figure 9-3).

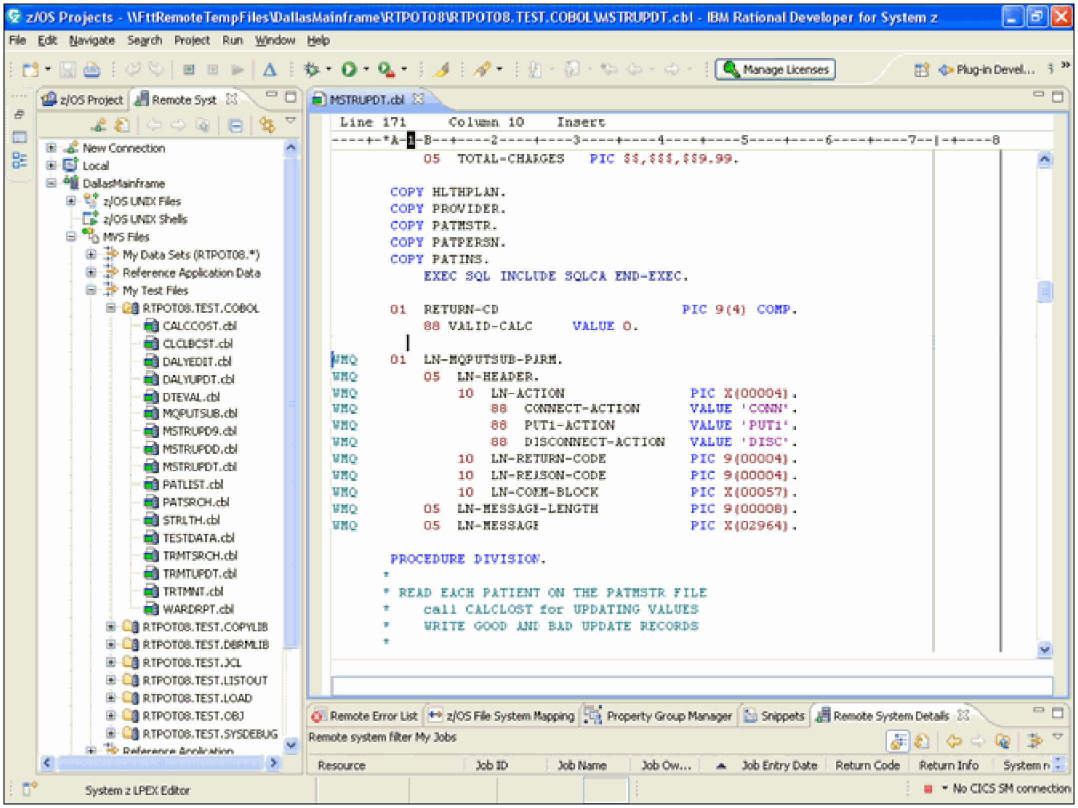


Figure 9-3 Rational Developer property groups: WebSphere MQ copy library concatenated with SYSLIB

Updating the property group that is associated with the program to include the SCSQCOBC library in the copy libraries enables the developer to hover over the WebSphere MQ-supplied working storage fields in the same way that we described in “Content Assist and hover” on page 113.

See Figure 9-4 for an example of hovering over one of the supplied MQ constants, MQPER-PERSISTENT. MQPER-PERSISTENT is moved to the MQ message descriptor field MQMD-PERSISTENCE. It requests that the message being put is marked as a persistent message, which is required in this instance to ensure that the message is not lost before being delivered to its destination application. Seeing the name of the constant and where it fits can help the developer understand the flow of the program better.

Figure 9-4 also shows the MQPUT1 API call, which has eight parameters:

- WS-CONNECT-HANDLE: The connection handle returned by WebSphere MQ after a successful connection. The handle uniquely identifies the connection between the application, in this case, the batch job running MSTRUPDT, and the queue manager.
- MQOD: MQ object descriptor. It describes the WebSphere MQ object that will be the subject of the MQ call. In this case, the object is a queue. The queue name is moved to MQOD-OBJECTNAME.
- MQMD: MQ message descriptor. It describes the WebSphere MQ properties of the message. The MQMD travels the Websphere MQ with the message.
- MQPMO: MQ put message options. It describes the required options on the put operation. In this case, we request that the operation is not part of the unit of work and that, if the queue manager is stopped while the operation is in flight, the MQPUT1 must fail rather than hold up the shutdown of the queue manager.
- WS-BUFFER-LEN: The length of the message.
- WS-MSG-BUFFER: The message. In this case, the patient master file record that has been written to the patient master file.
- WS-COMPCODE: This field is populated by WebSphere MQ with the completion code, normally either: 0 (`MQCC-OK`), 1 (`MQCC-WARNING`), or 2 (`MQCC-FAILED`).
- WS-REASON: This field is populated by WebSphere MQ with the reason code that is associated with the completion code. For example, `2085 - MQRC-UNKNOWN-OBJECT-NAME` typically indicates that the wrong queue name has been specified or that the queue name has not been defined.

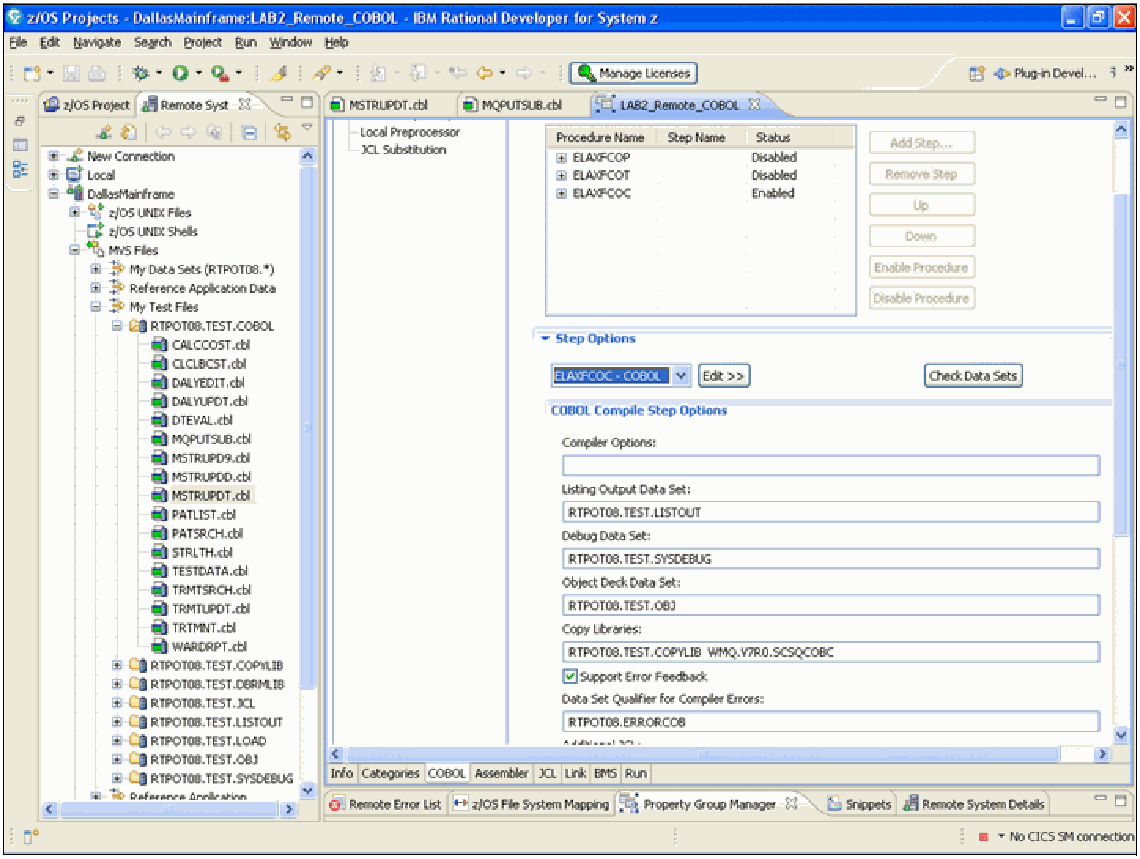


Figure 9-4 Rational Developer hover help over the WebSphere MQ fields

9.3.3 Required WebSphere MQ objects

WebSphere MQ provides the RUNMQSC utility to define, list, and manipulate WebSphere MQ objects and their properties. On z/OS, the CSQUTIL program provides this functionality. Example 9-2 is a sample of JCL to run CSQUTIL against the WMQA queue manager to define a remote queue. The QREMOTE definition maps an entity on the local queue manager to a target queue on another queue manager. The data warehouse application monitors the target queue. The data warehouse application gets the message from the queue and applies the information to the data warehouse for later analysis.

Example 9-2 JCL to run CSQUTIL against WMQA to define a remote queue

```
//S010 EXEC PGM=CSQUTIL,PARM='WMQA'
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
 COMMAND
/*
//CSQUCMD  DD *
DEFINE QREMOTE ('PRODZOS.DATA.WAREHOUSE.FEED') +
 RNAME ('PRODWHS.PATIENT.UPDATE.DATA') +
 RQMNAME (PRDWHS1)
 /*
```

To enable the transportation of the message from WMQA to the PRDWHS1 remote queue manager, you must define a WebSphere MQ channel. The application does not need to know the details of this definition. See Example 9-3 for a sample WebSphere MQ channel definition.

In this case, we chose the TCP/IP transport type. The conname field must resolve to the IP address on the TCP/IP stack with which the remote queue manager is associated. If the remote queue manager listens on the default port of 1414, you do not need to specify the conname field. The name of the channel is arbitrary, but we suggest that you follow a naming standard. The XMITQ name must be the same name as the target queue manager name. The other settings are outside the scope of this document, but they center around whether to stop the channel after a period of inactivity and how often to retry the connection if the channel stops abnormally.

Example 9-3 Sample WebSphere MQ channel definition

```
DEFINE CHANNEL (WMQA/PRDWHS1) +
 SEQWRAP(999999999) +
 CHLTYPE(SDR) +
 CONNAME (PRDWHS1) +
 XMITQ (PRDWHS1) +
 DISCINT(600) +
 SHORTRTY(1200) +
 SHORTTMR(5) +
 LONGRTY(999999999) +
LONGTMR(60) +2
 TRPTYPE(TCP)
*
DEFINE PROCESS(PRDWHS1) +
        DESCR( 'PROCESS TO TRIGGER WMQA/PRDWHS1') +
        APPLTYPE(MVS) APPLICID('CSQX START') +
        USERDATA('WMQA/PRDWHS1')
```

```
*
DEFINE QLOCAL (PRDWHS1) +
 DESCR ('TRANSMISSION QUEUE TO PRDWHS1') +
 INITQ('SYSTEM.CHANNEL.INITQ') +
 PROCESS ( 'PRDWHS1' ) +
 TRIGGER +
 TRIGTYPE( FIRST ) +
 TRIGMPRI( 0 ) +
 TRIGDPTH( 1 ) +
 USAGE (XMITQ)
```

We defined the process to enable the channel to start automatically when the first message arrives on the transmission queue. We defined the channel to stop after 10 minutes of inactivity (DISCINT(600)). Therefore, without defining this process and without defining the transmission queue to trigger on the arrival of the first message, we must start the channel manually.

On PRDWHS1, we need to define the target of the remote queue definition, PRODWHS.PATIENT.UPDATE.DATA. The warehouse application needs to monitor the arrival of messages on that queue so that the warehouse application can pick up and process the messages. The channel type on WMQA is a sender. For this channel to operate, it must have a identically named receiver channel on PRDWHS1. Example 9-4 is an example of the receiver channel definition.

Example 9-4 Defining the identically named receiver channel on PRDWHS1

```
DEFINE CHANNEL (WMQA/PRDWHS1) +
 SEQWRAP(999999999) +
 CHLTYPE(RCVR) +
 TRPTYPE(TCP)
```

9.4 WebSphere MQ debug example

Working with WebSphere MQ means specifying many values in the API parameters. Enhancing the visibility of these settings during analysis, coding, debugging, and testing is perhaps even more valuable to a developer than when working with more standard programs. To illustrate this point, consider the following example. We added the WebSphere MQ subprogram called MQPUTSUB to the MSTRUPDT program. In testing, the target queue, which in production is a remote queue, is defined as a local queue. Defining the target queue as a local queue in testing allows the developers who maintain the application to view the messages put to the queue after they make changes to the application.

Using WebSphere MQ to enable the transfer of information between applications lends itself to this sort of regression testing, because it means that the program that supplies the message is loosely coupled to the program that accepts the message. As long as the message remains the same across application changes, downstream processing is unaffected.

To continue our example, the developer has been running tests, but the developer is not getting a display from MQPUTSUB, which indicates that no messages are being put to the local test queue. The developer submits the test job through Rational Developer, specifying a workstation IP address so that a debug session starts, which happens automatically. The developer steps through the program and interrogates the WebSphere MQ return code and reason code. The developer can also use Debug Tool to see that the message has been put as expected. See Figure 9-5.

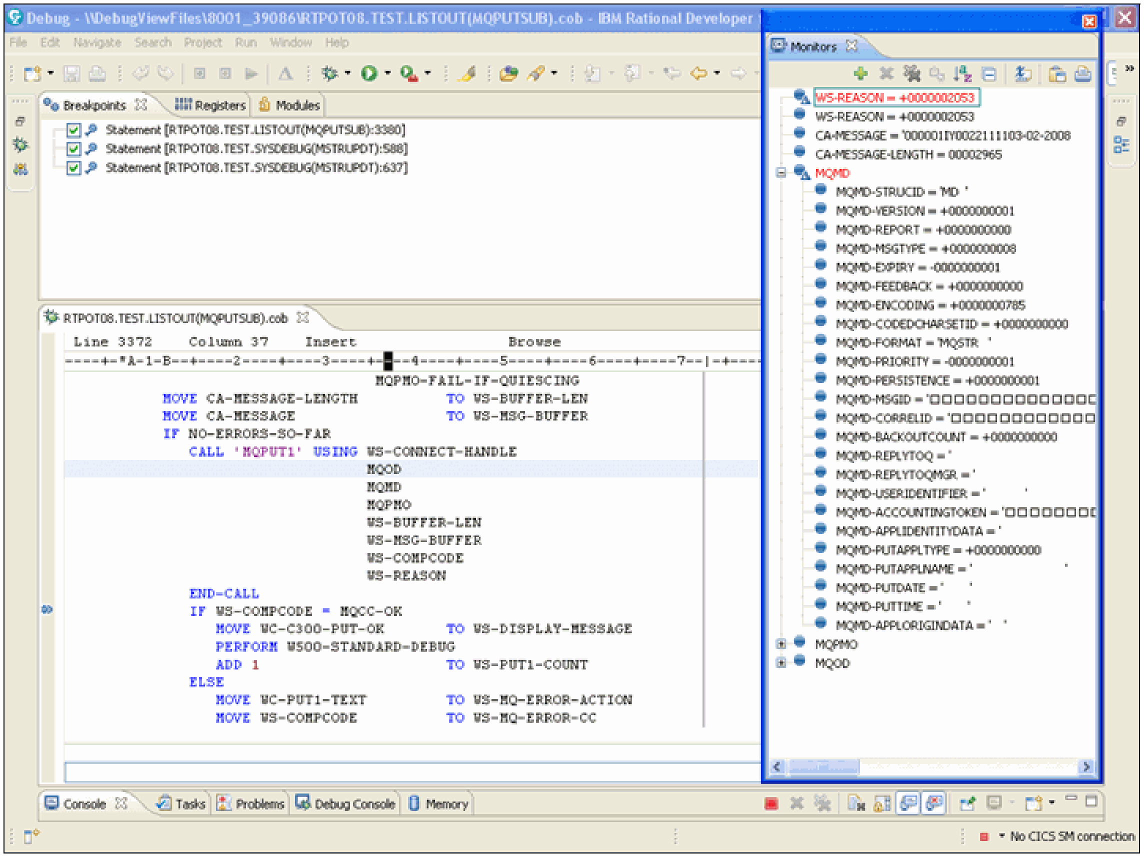


Figure 9-5 A live Debug Tool session showing WebSphere MQ field monitor values

Debug Tool shows the reason code as `2053 - MQRC_Q_FULL`. The developer had defined the queue to have a maximum depth of 1,000 messages. After a number of test runs, the queue reached this maximum. The subsequent run resulted in the 2053 ABEND. The developer cleared the queue and ran the job again successfully.

9.5 Comparisons between developing with Rational Developer and ISPF

WebSphere MQ programs do not differ from normal programs. There are several supplied copybooks that must be included. The documented API calls do not differ from a call to another subprogram. There are also extra requirements when preparing the programs that are detailed in the WebSphere MQ product documentation. From that point, working with these programs with tools, such as Rational Developer and Rational Asset Analyzer, does not differ from working with the tools with standard programs as we outlined earlier in this book. Likewise, all the advantages of the tools of the optimized life cycle over the traditional approach using ISPF that have been previously listed are equally applicable when working with WebSphere MQ-enabled programs.

9.6 Summary

WebSphere MQ-enabled programs do not differ that much from standard programs, so all the benefits previously outlined in this book are still true. The way in which the WebSphere MQ interface is implemented means that it is important to understand and correctly specify its many settings and properties to achieve the desired results. The way that the tools of the optimized life cycle allow the developer to view storage definitions within the context of the program flow during analysis, development, debugging, and testing greatly enhances and accelerates these processes, leading to better outcomes more quickly.

See the following references for further information about WebSphere MQ:

- MQ Primer:

  http://www.redbooks.ibm.com/abstracts/redp0021.html?Open

- WebSphere MQ V7.0 features and enhancements:

  http://www.redbooks.ibm.com/abstracts/sg247583.html

- IBM WebSphere MQ library:

  http://www-01.ibm.com/software/integration/wmq/library/

10

Business rules

In this chapter, we describe products and a methodology that can help you implement business rules. Even if you are already familiar with the tools that we describe in this chapter, reviewing these tools from a business perspective can help you communicate with the business user for whom you are developing and maintaining your applications.

This chapter contains the following topics:

- Business policy, business rules, business rule management systems
- IBM WebSphere ILOG BRMS architecture
- Using IBM WebSphere ILOG BRMS
- Business rule mining
- Governance
- The benefits of using WebSphere ILOG BRMS

The examples that we use in this section reference a loan application, because the documentation and training materials for these products primarily refer to a loan application.

10.1 Business policy, business rules, business rule management systems

First, we define the important terms that integrate your business and your applications:

- Business policy:

  Every organization has people who set the policies by which the organization does business. A *business policy* is a statement of guidelines governing business decisions. For example, a bank might have a lending policy that says, "*Customers whose credit rating is above average are entitled to a discounted rate on their loan*".

- Business rules:

  The specific statements that enforce a policy are *business rules*. Policies are translated into business rules, which are the detailed conditions and actions that unambiguously enforce the policy. The business rules expand upon the policy by stating in detail the circumstances under which the policy is applicable and the actions that enforce it. A policy can be translated into many business rules. In the bank loan policy described in the previous bullet, for example, the rules need to define the terms of the policy (for example, *customer, discounted rate,* and *loan*). Regional regulations can require that rules vary from region to region or that the discounted rate change over time as interest rates and economic conditions fluctuate.

A well-written business rule is a statement of business logic with these characteristics:

- Is not cryptic code
- Can be authored by business users
- Is self-documented, executable, and reusable

A business policy can require several, even hundreds, of business rules to express it. Business logic usually requires a series of smaller decisions to be made before the final result can be determined. For example, when determining the rate for a customer's loan, the key decision might be the final rate applied to the loan. To make that decision, a series of smaller decisions might have to be made, such as these decisions:

- Has the customer provided all the requested identifying information?
- What is the total of the customer's current income?
- What is the amount of the loan the customer is applying for?

Each of these smaller business decisions might also require several business rules to implement the logic. With hundreds or thousands of rules being used to implement business policies, and with ongoing changes to the details that support the policies, organizations need a system to manage and update rules efficiently, thus, the need for a *business rule management system* (BRMS).

10.1.1 Business rule management system

BRMS is a system for managing and storing business rules independently from the application source code. In existing systems, business policies can be hard-coded in the application, making it impossible for business users to maintain or update the rules without help from the development team. Changes can put the stability of the system at risk, and testing and deployment cycles can take a long time. By externalizing the business logic from a business application with business rules, business users can manage business policies directly with limited dependence on the IT department. The degree of dependence can range from limited review by business users of policies implemented by developers, to complete control over the specification, creation, testing, and deployment of the rules by business users.

10.1.2 Advantages of a business rule management system

Using a BRMS provides the following advantages:

- Improves the agility (and therefore reduces the time, effort, and cost) that is required to develop and maintain applications by separating business decision logic from core application code:
  - Business decisions and rules can be accessed and changed more easily.
  - Business decisions and rules can be reused across applications.
- Provides better management of rule-based decisions:
  - Rules are stored and managed in a centralized repository, with comprehensive capabilities for querying and reporting on rules and associated metadata.
  - Rules can be defined in a customizable business vocabulary using a variety of rule metaphors, including decision tables, decision trees, scorecards, and text-based rules.
  - Tools and environments are designed specifically for the unique needs of various stakeholders involved in rule management: line of business personnel, business analysts, systems developers, and IT operations. Business users can take responsibility for changing the rules that support their businesses.

- Establishes a road map for incremental application modernization:
  - Decision logic can be defined with increased granularity and precision, increasing automation in business systems and ensuring that systems make the right decision based on the context and requirements of each customer, transaction, or activity.
  - Governance processes around business rules allow users to understand how and where rule changes affect business decisions.
  - Rules that are written to support existing applications can be shared with applications that use leading-edge technology, providing a path for the application portfolio to be modernized over time.

10.2 IBM WebSphere ILOG BRMS architecture

IBM WebSphere ILOG BRMS includes a set of modules that work together across business and development environments to provide a comprehensive solution for managing business rules. The ILOG BRMS modules are aimed at specific user roles, based on their varied skill sets. ILOG's BRMS synchronization mechanisms allow developers and business users to collaborate on the same project, while working in their own environments and at their own pace.

The ILOG BRMS modules address both COBOL and distributed environments. In this book, we describe the following rule development and rule management modules:

- Rule Studio: Developer tool that supports both COBOL and distributed applications
- IBM WebSphere ILOG Rule Team Server: Business analyst and business manager tool that is used for the management of business rules throughout the rule life cycle that supports both COBOL and distributed rule applications
- IBM WebSphere ILOG Rule Solutions for Office: Business analyst and business manager tool for authoring rules that support both COBOL and distributed applications
- IBM WebSphere ILOG Decision Validation Services: Rule testing and simulation tool that supports both COBOL and distributed applications
- IBM WebSphere ILOG Rules for COBOL: Allows rulesets managed in WebSphere ILOG JRules or Rule Team Server to be generated into COBOL copybook code for use inside COBOL applications

IBM WebSphere ILOG JRules for z/OS contains two rule execution mechanisms for applications:

- Rules Execution Server (RES): Used to execute, manage, and monitor deployed business rules in the Java Rule Engine. The RES also provides the ability to deploy rules as a decision service for a service-oriented architecture (SOA) strategy that is known as *Transparent Decision Services*.
- IBM WebSphere ILOG Rules for COBOL: Generates Enterprise COBOL code from the authored and managed business rules.

Figure 10-1 illustrates the ILOG BRMS architecture. Development of a business rule application starts in Rule Studio. After setting up the rule authoring environment in Rule Studio, the rules are published to Rule Team Server. Business users write or maintain rules using Rule Team Server, and the rules are stored in the Rule Team Server repository. Developers can connect to Rule Team Server from Rule Studio to synchronize rules that are stored across environments.

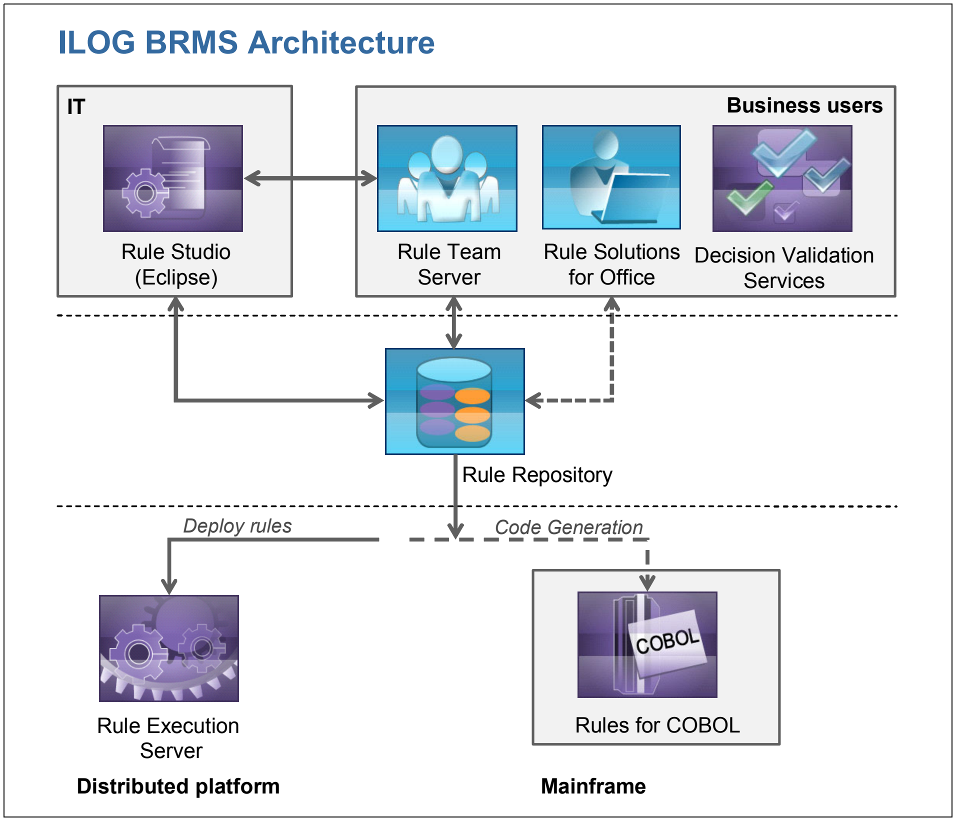


Figure 10-1 ILOG BRMS architecture

Rule Studio is an Eclipse-based product that provides a comprehensive development environment for rule-based applications. In Rule Studio, you can perform these tasks:

- Define the data (object model) on which the rules are based
- Use natural language to develop rules using terminology that is familiar to your business partners
- Define logic and flow to control the execution of business rules
- Choose the editor that is most compatible with the function on which you are working
- Ensure that the rules that you have authored are complete and consistent
- Synchronize rules with Rule Team Server

Rule Team Server is a tool to write and maintain business rules, both during application development and after the application is deployed. It also provides rule management features, such as versioning, baselines, and access control. In Rule Team Server, you can perform these tasks:

- Author and edit rules using a natural language customizable business vocabulary. Rules can also be expressed in graphical formats, such as decision tables and decision trees.
- Simplify rule projects through Smart Views (user configurable navigation panes), filters, and reports.
- Manage collaboration with role-based permissions, access controls, and customizable rule metadata properties.
- Facilitate rule maintenance with templates, point-and-click editors, error checking, and versioning.
- Ensure rule quality with customizable queries, rule analysis, and visual comparisons of changes between versions.

Rule Solutions for Office provides business users guided rule authoring and editing in Microsoft Office Word and Excel. Non-technical business experts can maintain business rules using familiar Microsoft Office desktop tools with automated guidance that ensures changes meet the underlying application requirements of the systems that will use these rules in the production environment. With Rule Solutions for Office, you can perform these tasks:

- Take advantage of specialized tool bars for Microsoft Office Word and Excel, that allow users to open viewing panes and lists that are specific to rule authoring and validation
- Get assistance from the Intellirule auto-completion technology, which can highlight problems, such as incorrect rule syntax for natural language rules, and gaps or overlaps in decision tables

- Work with “ruledoc” files, which can be downloaded from BRMS and edited offline
- Synchronize changes with Rule Team Server: Completed rule changes can be uploaded easily to the rule repository for versioning, analysis, and deployment
- Combine both structured rules and unstructured information for documentation purposes

Decision Validation Services provides rule testing, simulation, and audit capabilities for the ILOG BRMS product line. With Decision Validation Services, you can perform these tasks:

- Create customizable scenarios and run them against changes to rules to verify correctness, to assess business impact prior to deployment, and to review actual execution results for deployed rules
- Implement changes faster while reducing the risk of errors due to change, with comprehensive rule testing capabilities
- Ensure rule changes meet business objectives by running simulations against either historical or test data and assessing the results against key performance indicators (KPIs) and what-if scenarios
- Input data from either Microsoft Office Excel or enterprise data sources
- Modify test cases and expected outputs easily and send results to Excel or HTML
- Audit rule execution results to review decision outputs

Rules for COBOL contains the core Rule Studio functionality and also allows rulesets to be generated into COBOL copybook code for use inside COBOL applications. With Rules for COBOL, you can perform these tasks:

- Add powerful BRMS functionality to existing COBOL-based applications to improve automated decisions
- Generate COBOL source code from ILOG BRMS rulesets for compilation into an executable format
- Generate COBOL copybook code that incorporates rule flows, natural language rules, decision tables, and decision trees
- Manage rules in a flexible, secure, and governed system that allows decision logic to be understood by both technical and business users
- Reuse business rules across multiple applications, systems, and processes
- Take advantage of Rules for COBOL and ILOG BRMS as a way to progressively modernize COBOL applications and accelerate the transition to an overall SOA strategy

10.3 Using IBM WebSphere ILOG BRMS

Before we present a scenario for developing a sample business rule project, let us define several IBM WebSphere ILOG BRMS-related terms and concepts:

- Rule project:

  Within Rule Studio, a *rule project* is a type of Eclipse project that enables you to logically manage, build, and debug the items that comprise the business logic of your application. When you define a rule project in Rule Studio, folders are created for Rules, business object models (BOMs), queries, and templates.

- Rule flow:

  In COBOL, we are familiar with many statements that control program logic, such as `if/then/else`, `do while`, and `for i=x to y by z`. In ILOG BRMS, the term for the logic that controls rule execution is *rule flow*.

- Rule package:

  Within a rule project, you organize rule artifacts into *rule packages*. Any application can have several aspects, such as business function (pricing or scoring), geography, product, responsibility, and so on. As a general guideline, you use a simple rule package structure to capture one of these aspects, and use custom rule properties to map other aspects. Rule packages are also a useful way of preparing rule flow tasks. Each rule package can correspond to a task. A *ruleset* is the end result of packaging the rule artifacts for execution.

- Business object model:

  The *business object model* (BOM) is the business view of the model that defines the actions and entities that are used in business rule artifacts. If you are familiar with object-oriented programming, think of the BOM as containing the classes and methods for your project. If you are familiar with COBOL, think of the BOM as your data files/data bases and the logic that is performed on that data.

- Verbalization:

  Verbalization is defining the vocabulary based on natural language to assist business users in understanding the business rules. Because most business users do not know COBOL, the term `ML_CUST_CD` will likely not be familiar, but "`the customer`" allow business users to immediately understand what the rule refers to.

10.3.1 A sample ILOG BRMS scenario for COBOL execution

Now that we have defined the terms and concepts, it is time to look at an example of using ILOG BRMS. In this sample, we start with a COBOL copybook, use it to create a BOM, develop rules that are based on the BOM, and generate COBOL code from the rules that we have written.

We use Rule Studio and Rules for COBOL for these tasks. You can use the same rule authoring steps if you are looking to deploy the business rules to the ILOG Rule Execution Server on z/OS. We do not document every step in the process, and we do not provide every screen shot, but the example shown in Figure 10-2 illustrates the general process that you follow.

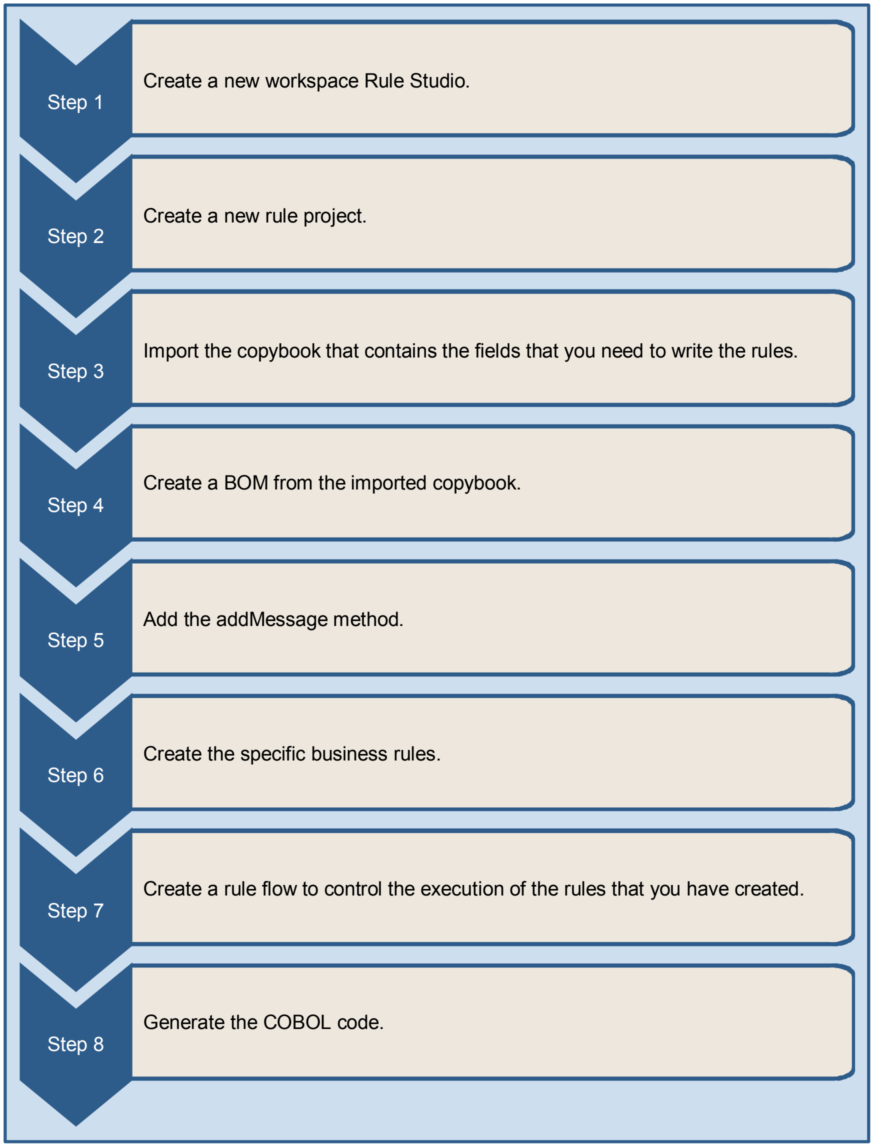


Figure 10-2 Process using ILOG BRMS

Step 1: Create a new rule project in Rule Studio

Follow these steps to create a new rule project in Rule Studio:

1. Start Rule Studio (Launch Rule Studio. Click **Start** → **All Programs** → **IBM WebSphere ILOG JRules V7.1** → **Rule Studio**.
2. On the Workspace Launcher dialog box (Figure 10-3), type a name for the file that you are creating (choose a directory and name that you will remember).
3. Click **OK**.

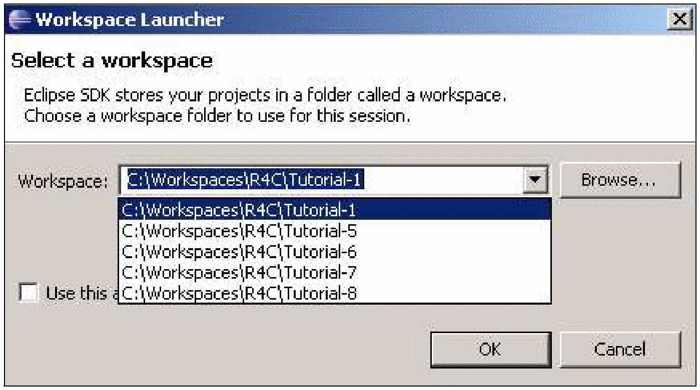


Figure 10-3 Selecting a workspace

Step 2: Create a new rule project

Follow these steps to create a new rule project:

1. On the File menu, click **New** → **Rule Project**.
2. Select **Standard Rule Project**, and click **Next**.
3. On the New Project page, in the Project name field, type `miniloan-rules`.
4. Make sure that the **Use default location** box is selected, so that your rule project is saved to your Eclipse workspace.
5. Click **Finish**.
6. If you are not currently in the Rule Perspective, a dialog box appears with the following message:

   ```
   This kind of project is associated with the Rule Perspective. Do you
   want to switch to this perspective now?
   ```

7. Click **Yes** to change to the Rule Perspective.

8. Your new rule project now appears in the Rule Explorer view of the Rule Perspective (Figure 10-4).

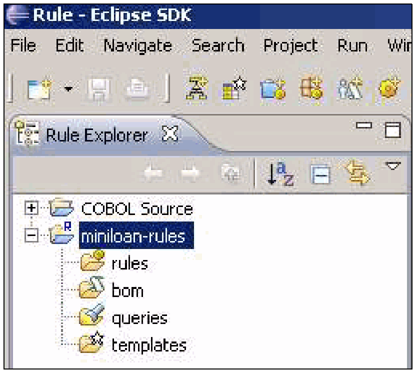


Figure 10-4 Displaying the new rule project in the Rule Explorer view

Notice the default folders (rules, bom, queries, and templates) that are created for your rules project.

Step 3: Import the copybook

Import the copybook that contains the fields that you need to write the rules. Example 10-1 shows the copybook data that we used in our scenario.

Example 10-1 Copybook data for our loan application scenario

```
01  Borrower.
           05  name                   PIC  X(20).
           05  creditScore            PIC  9(10).
           05  yearlyIncome           PIC  9(10).
           05  age                    PIC  9(3).
               88  teenager                     VALUE 0  THRU 17.
               88  adult                        VALUE 18 THRU 60.
               88  retired                      VALUE 61 THRU 150.
01  Loan.
           05  amount                 PIC  9(10).
           05  yearlyInterestRate     PIC  99.
           05  yearlyRepayment        PIC  9(10).
           05  effectDate             PIC  X(8).
           05  approved               PIC  X.
           05  messageCount           PIC  9(2).
```

Follow these steps to import the copybook:

1. On the File menu, click **Import**.
2. In the Import dialog box:
   a. Click **General** to expand.
   b. Click **File System**.
   c. Click **Next**.
   d. In “From Directory,” click **Browse** to navigate to the file that has the copybook, and select it.
   e. In “Into Directory,” click **Browse** and navigate to the folder in which you want to save the copybook (navigating to the folder selects it).
   f. Click **Finish**.

Step 4: Create a BOM from the imported copybook

This step creates the business object model (BOM) that is needed to author the business rules. The source of the COBOL copybook identifies the mainframe fields in which the rules need to be written. After the import is done, you can define the business terms and use them in the rule authoring instead of COBOL field names, for example:

1. In the Rule Explorer view, right-click the bom folder in the miniloan-rules project, and select **Rules for COBOL** → **Create BOM from COBOL Copybook** (Figure 10-5 on page 450).

   When you import the BOM, Rules for COBOL verifies that it is importing a valid COBOL copybook.

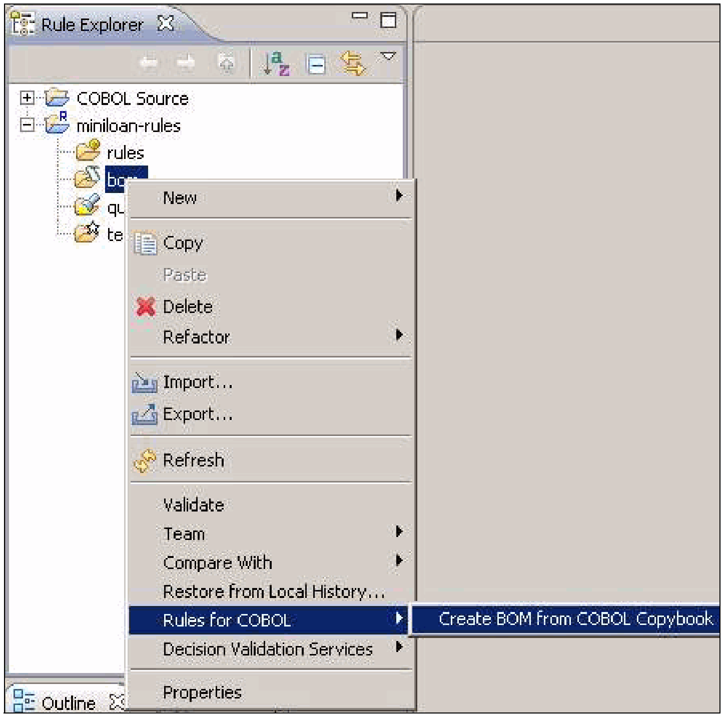


Figure 10-5 Creating the BOM from the COBOL copybook

2. In the New BOM Entry wizard, click **Browse** next the Copybook field and navigate to the miniloan copybook.
3. Browse to the directory and file that have your copybook source.
4. Select **miniloan.cpy**, and click **Open**.
5. In the Package field, type `miniloan`.

6. In the BOM entry Name field, leave the default value `model` (Figure 10-6).

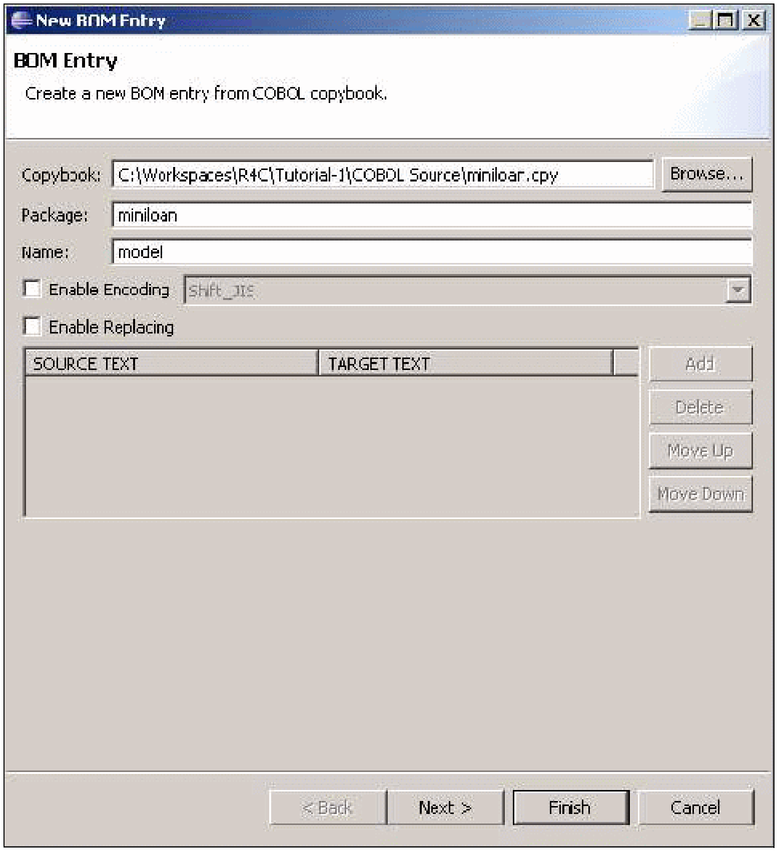


Figure 10-6 Creating a new BOM entry from the COBOL copybook

7. Click **Next**.

8. On the COBOL items selection page, leave all items selected, and click **Next** (Figure 10-7).

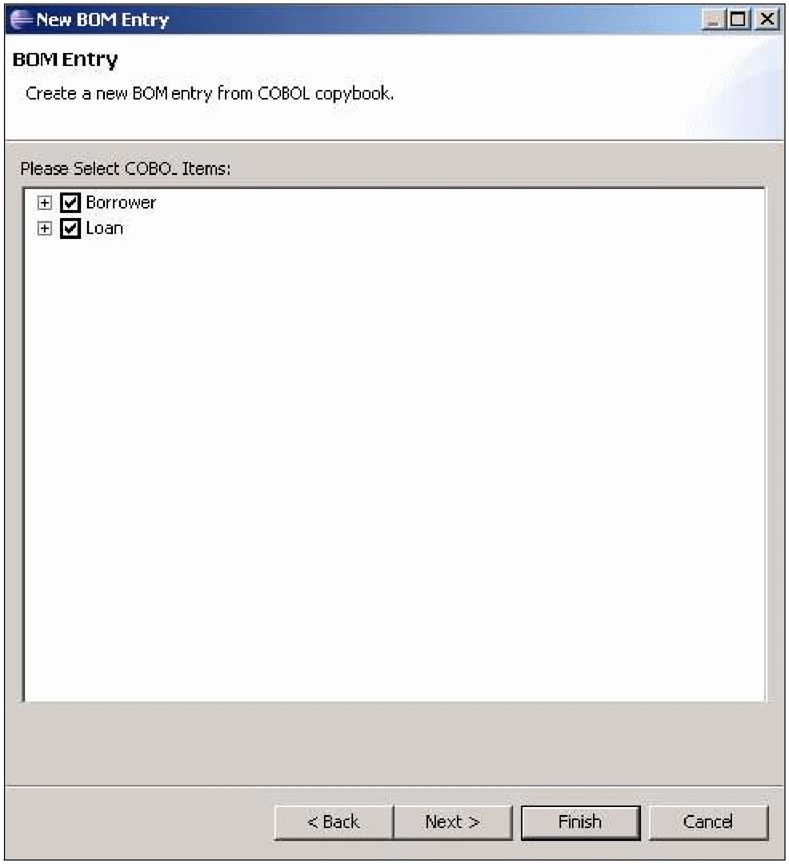


Figure 10-7 Selecting the COBOL items

9. On the Verbalize the ruleset parameters page (Figure 10-8), verbalize the COBOL data items:

   a. On line one, click in the Verbalization column next to the Borrower item, and type `the borrower`.

   b. Click in the Verbalization column next to the Loan item, and type `the loan`.

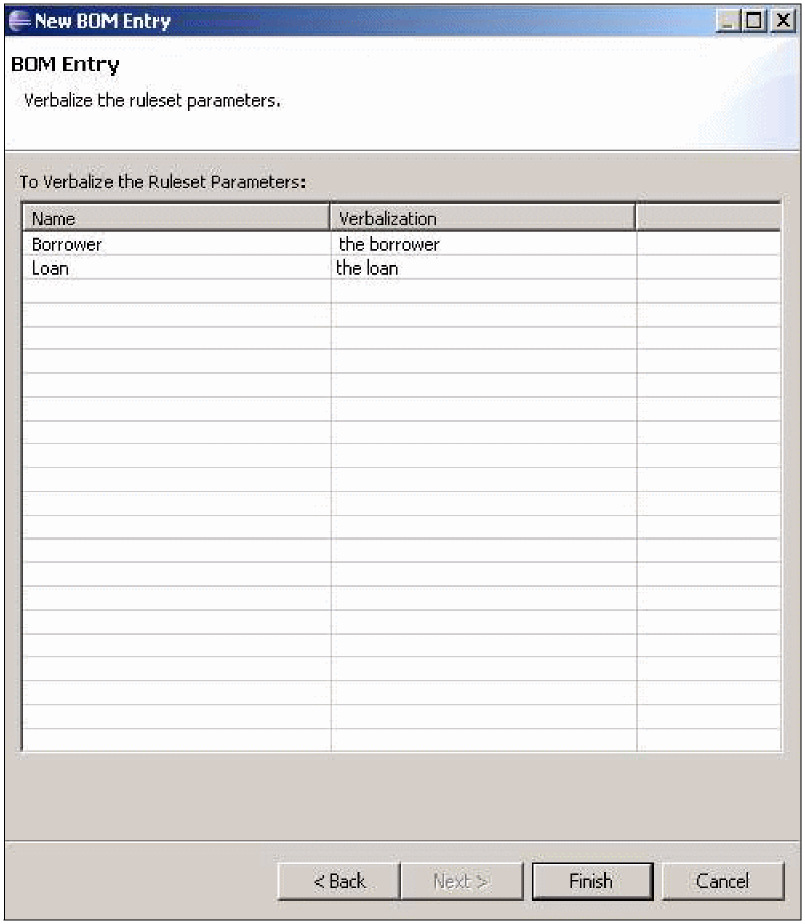


Figure 10-8 Verbalizing the ruleset parameters

10. Click **Finish**.

    A new BOM entry appears under the bom folder in the Rule Explorer view. The bom folder needs to look like Figure 10-9.

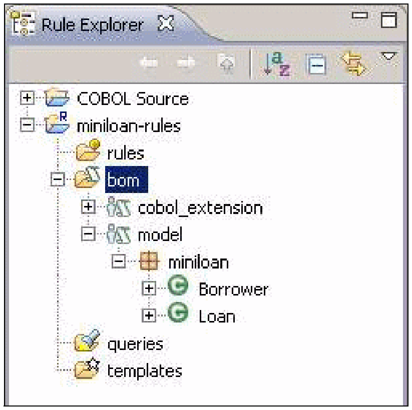


Figure 10-9 Displaying the bom folder

You now have a package named `miniloan` that contains two classes: `Borrower` and `Loan`.

In the next several steps, we write the COBOL code for the BOM methods.

Rules for COBOL provide a BOM method mapping to COBOL code to enable you to define actions for your rules. When you generate the BOM from the COBOL copybook, all the BOM classes that are created contain only fields (attributes). You need to add the methods to pass information between the COBOL subprogram that executes the rules and the calling COBOL program. You use *methods* to author rules and specify conditions and actions.

You create methods in Rule Studio and then implement them in COBOL, so that, when you generate the COBOL program to execute the rules, the generated COBOL code will invoke the methods (Table 10-1).

Table 10-1 BOM method

Method	Description
`public static void addMessage(String msg)`	Defines what is needed to pass information from the rules

Step 5: Add the addMessage method

In this step, we add the addMessage method to the Loan class. This method is called to add a message to the loan object when a business rule needs to add (error) information. The method takes a string (containing the message) as a parameter.

Follow these steps to add the addMessage method:

1. In the Rule Explorer (top left panel) double-click the **miniloan** package in the model, which will open the Outline view (lower left panel).
2. In the Outline view, expand the BOM entry: model and click the **Loan** class.

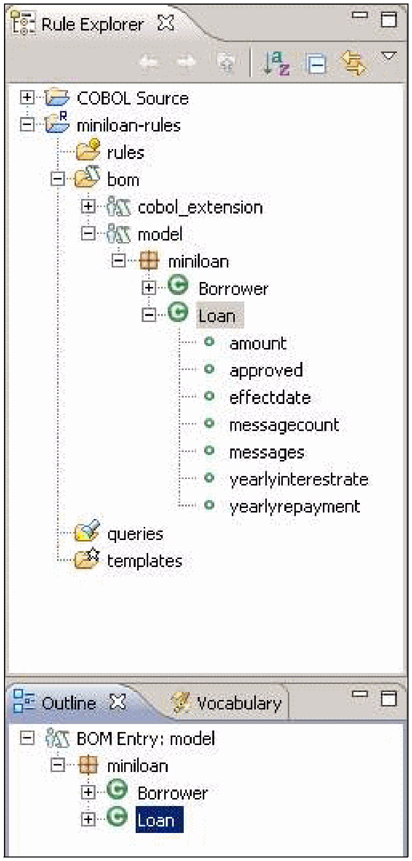


Figure 10-10 Selecting the Loan class

3. On the Class page of the BOM editor, in the Members section, click **New** (Figure 10-11).

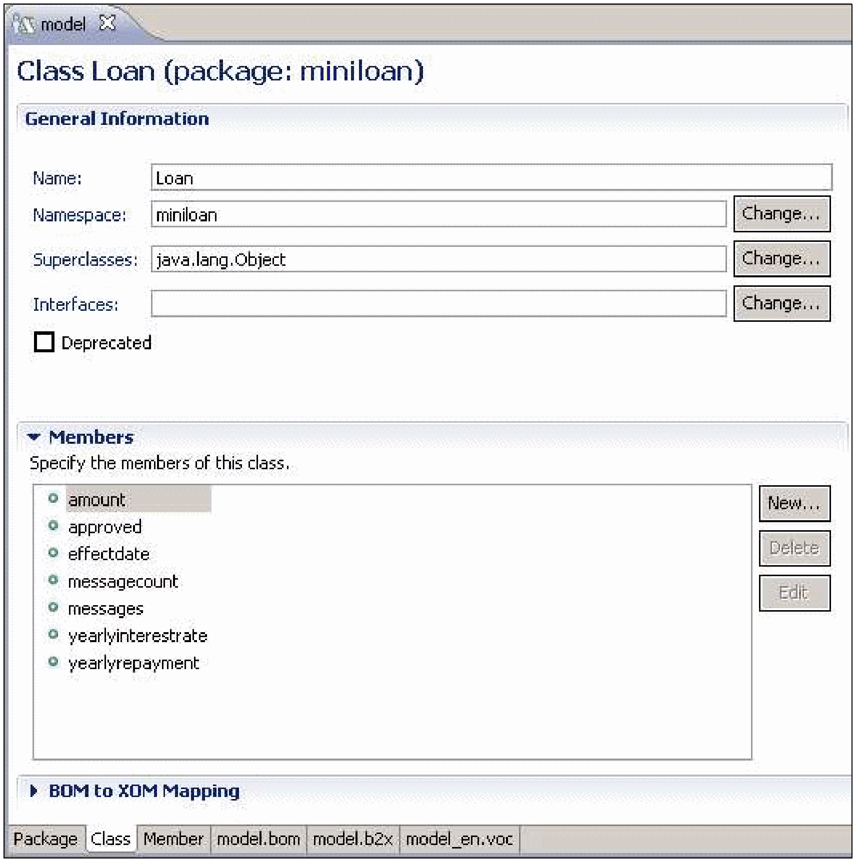


Figure 10-11 Specifying the class members

4. In the New Member dialog box on the Member page, select the type **Method**. In the Name field, type `addMessage`. In the Type field, type `void` (Figure 10-12).

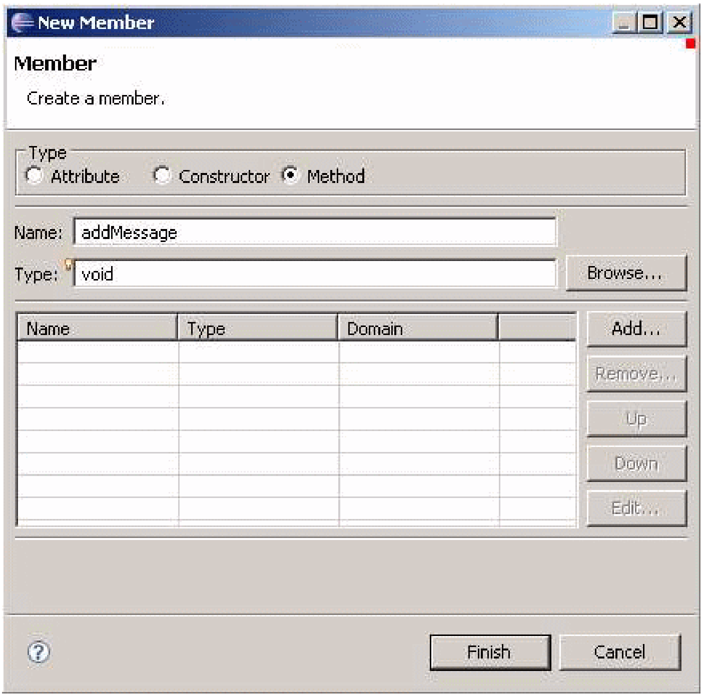


Figure 10-12 Creating the member

In Figure 10-12, click **Add**.

5. In the Name field of the Method Argument, type `msg` (Figure 10-13).

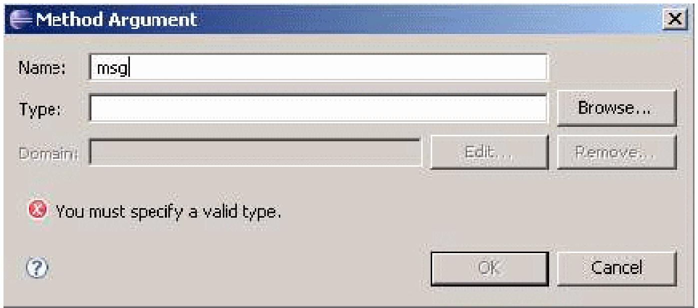


Figure 10-13 Creating the method argument

Select **Browse** next to the Type field.

6. In Figure 10-14, in the Types panel, type `str`, and select **String** (**java.lang.String**).

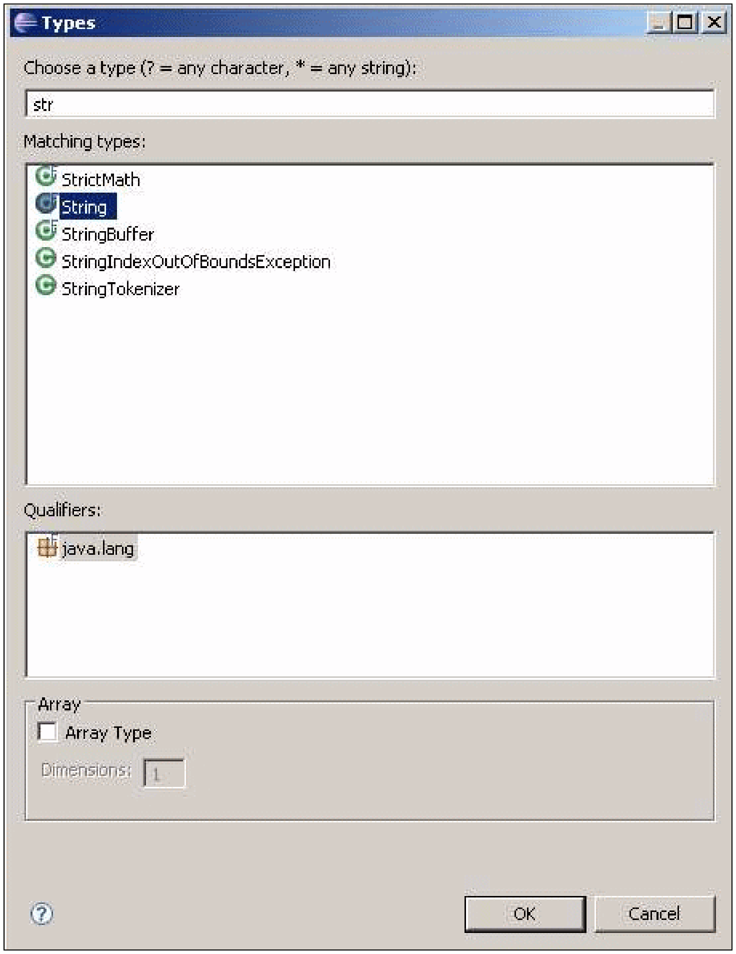


Figure 10-14 Selecting the type

Click **OK** to close the **Types** dialog.

7. Click **OK** to close the **Method Argument** dialog.

8. On the New Member page, click **Finish** (Figure 10-15).

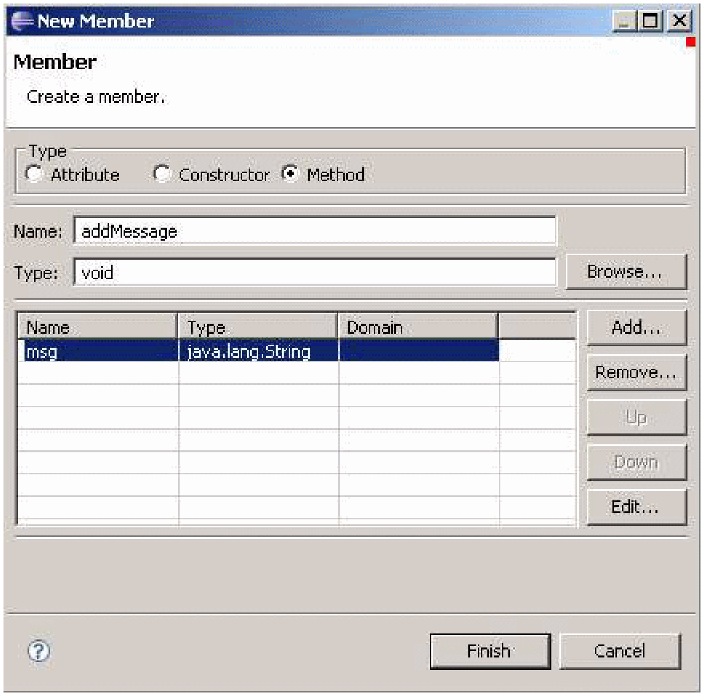


Figure 10-15 Clicking Finish to create a member

You return to the Class Loan page of the BOM editor, where the addMessage method has been added to the Members list (Figure 10-16).

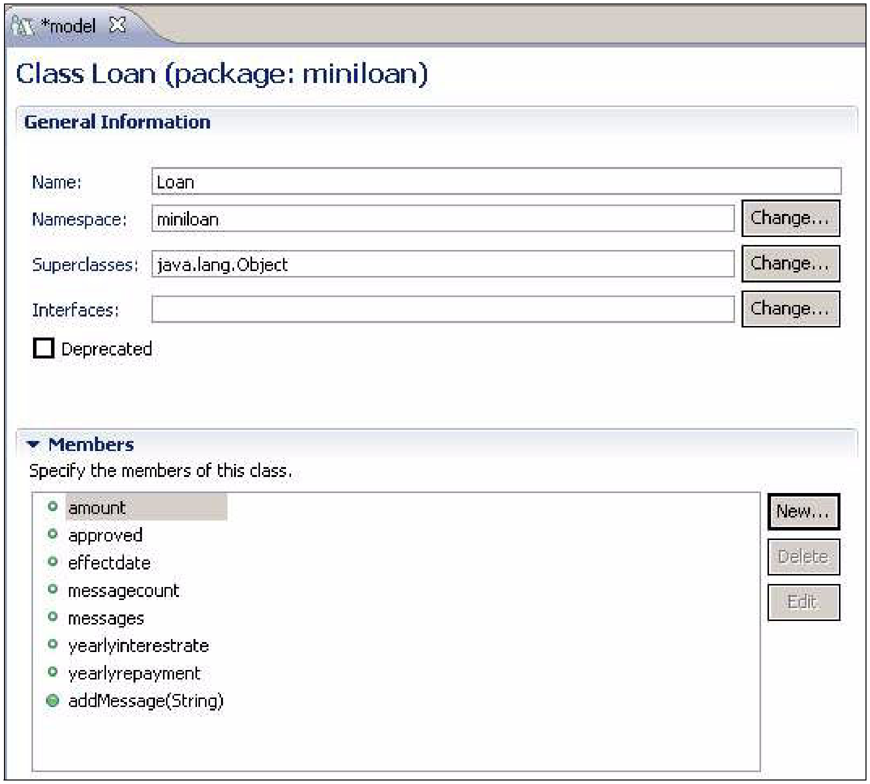


Figure 10-16 Adding the addMessage method to the members list

9. In Figure 10-16, double-click the **addMessage** member.
10. On the Member page of the BOM editor, under General Information, select **Static** (Figure 10-17).

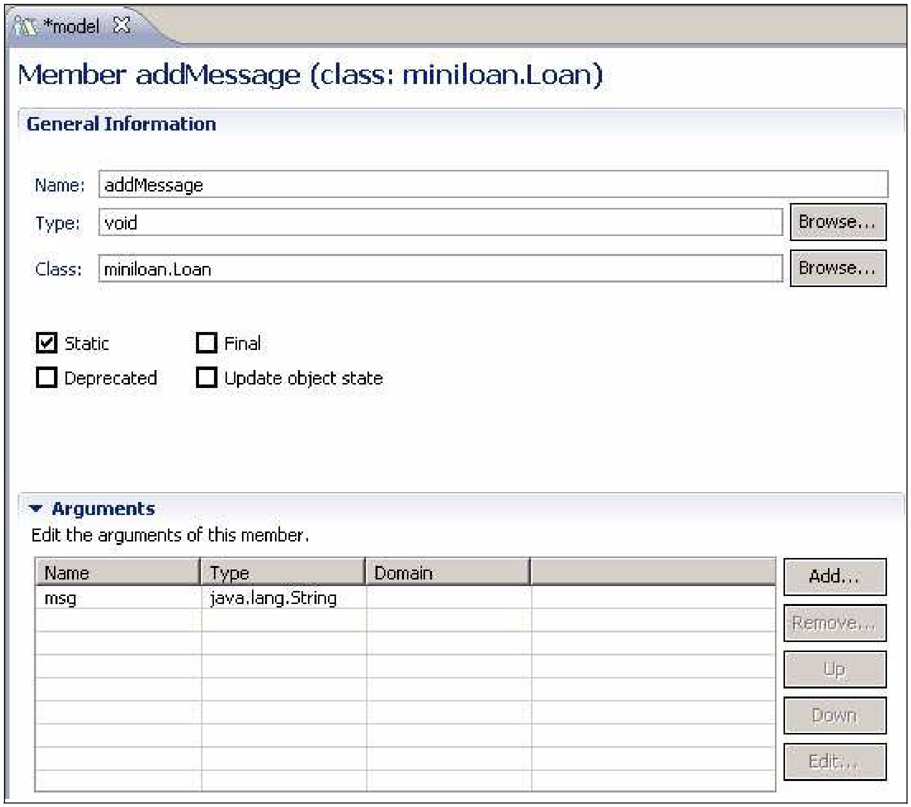


Figure 10-17 Selecting static for the addMessage method

11. Verbalize the new method:
    a. In the Member Verbalization section of the BOM editor, click **Create**.
    b. Under Action, edit the Template field to read: `add message {0}`.
    c. Scroll down to the COBOL Method section of the BOM editor, and click **Create a mapped method**. This option generates the data type definitions for the method and shows them in the Mapped Method table.
12. In the COBOL Code Body editor, add the following COBOL code to implement the method:

```
COMPUTE MESSAGECOUNT = MESSAGECOUNT + 1
MOVE {msg} TO MESSAGES(MESSAGECOUNT)
```

13. Change the COBOL Picture from X(20) to `X(60)` to bring it in-line with the COBOL copybook.
14. Save your work by typing Ctrl+S.

You can now use this method to create actions in your rules.

Step 6: Create the specific business rules

You use natural language, Content Assist, and pointing and clicking to create the rules. We do not document the specific steps here.

Step 7: Create a rule flow

To control the execution of the rules that you created, you need to create a rule flow. We do not describe the specific steps in this book.

Step 8: Generate the COBOL code

Follow these steps to generate the COBOL code:

1. In the Rule Explorer, right-click the **miniloan-rules** project and select **Rules for COBOL** → **Generate COBOL Code** (Figure 10-18).

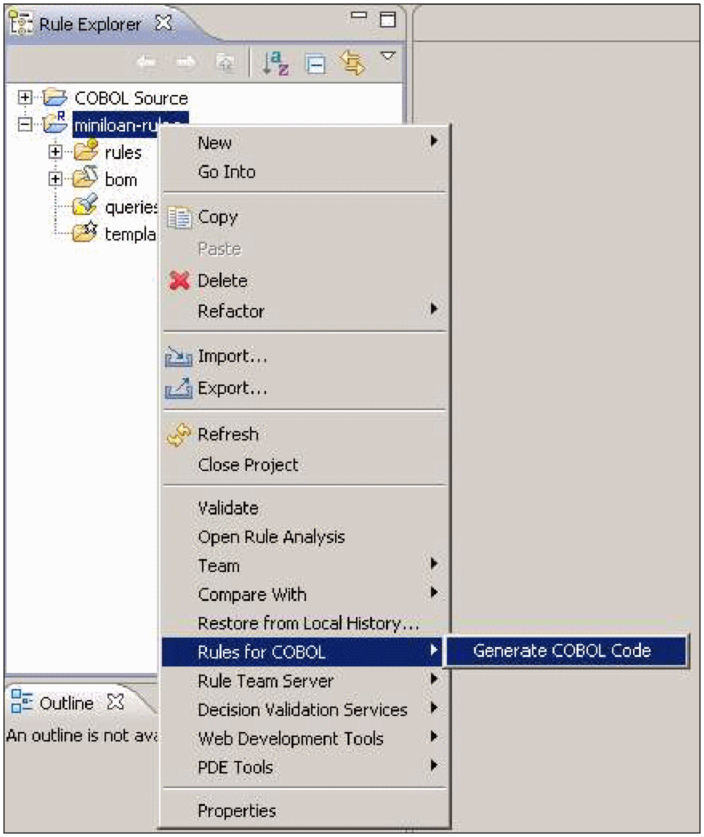


Figure 10-18 Selecting Generate COBOL Code

2. In the Generate COBOL Code wizard (Figure 10-19), in the COBOL Program ID field, type `minicbl`.
3. In the Output file field, type the name of the file in which you want to store the COBOL code, including its directory path. In this case, we chose `C:\?\COBOL Source\minicbl.cbl`. Replace the question mark in the path with the name of the workspace in which you are currently working.

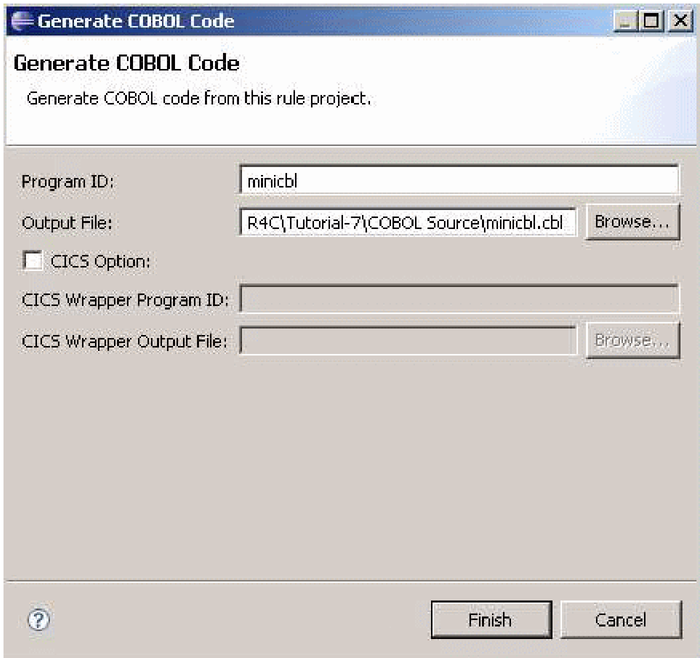


Figure 10-19 Generating COBOL code

Click **Finish**.

4. A COBOL subprogram is generated (Figure 10-20). It contains the ILOG BRMS ruleset converted to COBOL code.

Figure 10-20 COBOL code generated successfully

5. Click **OK**.
6. Open the **COBOL Source** project in the Rule Explorer, and click **Refresh** (Figure 10-21).

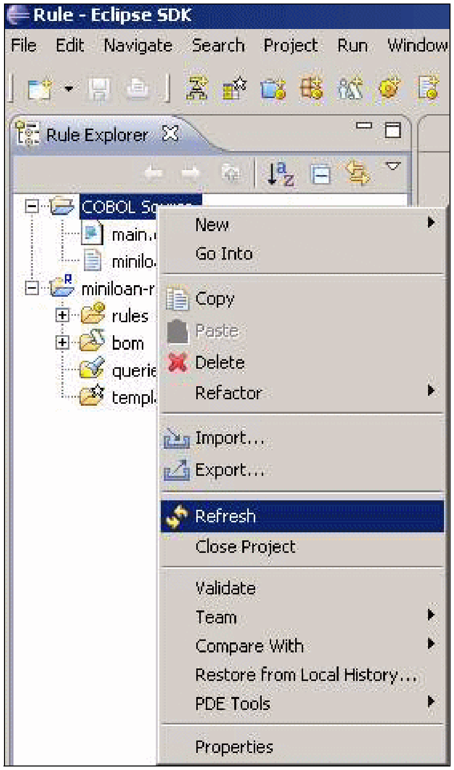


Figure 10-21 Refreshing the COBOL source code

7. Open the COBOL Source project. Right-click the generated **minicbl.cbl** file, and select **Open With** → **Text editor**.
8. You can review the code you have generated.
9. The code that is generated is a COBOL subprogram. In an actual scenario, you pass the COBOL subprogram to the System z application and compile it with the caller program. It is then ready for testing and execution.

10.3.2 ILOG BRMS features and functions

A complete description of all the features and functions of ILOG BRMS is beyond the scope of this book. However, we describe several of the many features and functions that make using ILOG BRMS fun and easy.

Decision tables and decision trees

Decision tables and decision trees provide a concise view of a set of business rules. You use *decision tables* when the rule logic is similar and repeating. For example, if the value of the requested loan is under $10,000, add 10 to the loan score. If the value of the requested loan is greater than $10,000 but less than $50,000, add 30 to the loan score. If the value of the requested loan is greater than $50,000, add 50 to the loan score.

Each of these cases has a minimum and maximum value to be tested and a specific value to use depending on the result of the test. It is easy to visualize how you can represent this rule in a table. You can populate the tables from Excel spreadsheets, and the tables are easy to maintain by business users. You use *decision trees* when the data is less symmetrical. Decision trees provide the same functionality as decision tables, but they are more adaptable to a set of rules that do not necessarily share the same set of conditions or actions.

Debugging

As anyone who has written a program knows, writing code that is syntactically correct is no guarantee that the code will run correctly. Rule Studio provides a debugger that allows you to step through the business rules, setting breakpoints as you want, to examine the behavior of the rules and to ensure that each test case produces a valid outcome. One of the advantages of running the debugger is that you can test iteratively, passing parameters and changing them so that all scenarios are covered in testing. This method, of course, does not eliminate the need for further testing after the COBOL code is generated, but it will dramatically reduce the need to revert back to Rule Studio or Rule Team Server after you have generated and compiled the COBOL code.

Rule Analysis

The Rule Analysis perspective in Rule Studio helps you identify potential problems within your ruleset, which can be especially useful if the rule project is complex. Running Rule Analysis alerts you to potential problems, but you are the final arbiter of whether the rules are correct. Rule Analysis will identify potential problems, but it will not solve them for you.

The two major types of analysis are Consistency Analysis and Completeness Analysis:

- Consistency Analysis identify problems with the following areas:
  - Rules that are never selected
  - Rules that never apply
  - Rules with range violations
  - Rules with equivalent conditions
  - Rules with equivalent conditions and actions
  - Redundant rules
  - Self-conflicting rules
  - Conflicting rules
- Completeness Analysis analyzes the condition parts of the rules in the ruleset and detects whether there are cases for which no rule is applicable. For each such detected case, the Rule Analysis view presents you with a new (missing) rule that can handle this case.

Queries

You can define your own queries to evaluate the impact of changes to the object model or changes to the rules. For example, before adding a new rule that modifies a certain data value to a rule project, you can run a query on all rules that use that parameter to assess the impact of this new rule. By default, queries are run on business rules, but you can also run them against other elements, such as rule flows, decision trees, and decision tables.

Templates

In certain cases, you have a number of business rules with similar content and structure. To support authoring them efficiently, ILOG BRMS lets you create a business rule template for this series of rules. A *business rule template* is a partially written business rule that is used to create multiple business rules with a similar structure. You can protect (freeze) parts or all of it to limit which parts of the rule can be edited. Writing rules that are based on templates makes it easier to author rules, because the structure is ready and you only have to modify the unfrozen parts of the rule.

Editors

In ILOG BRMS, you can use the following editors in Rule Studio to create the rule artifacts:

- Intellirule editor
- Guided editor
- Decision table editor
- Decision tree editor
- Code editor

Figure 10-22 illustrates the relationship between rule artifacts and editors that work on them.

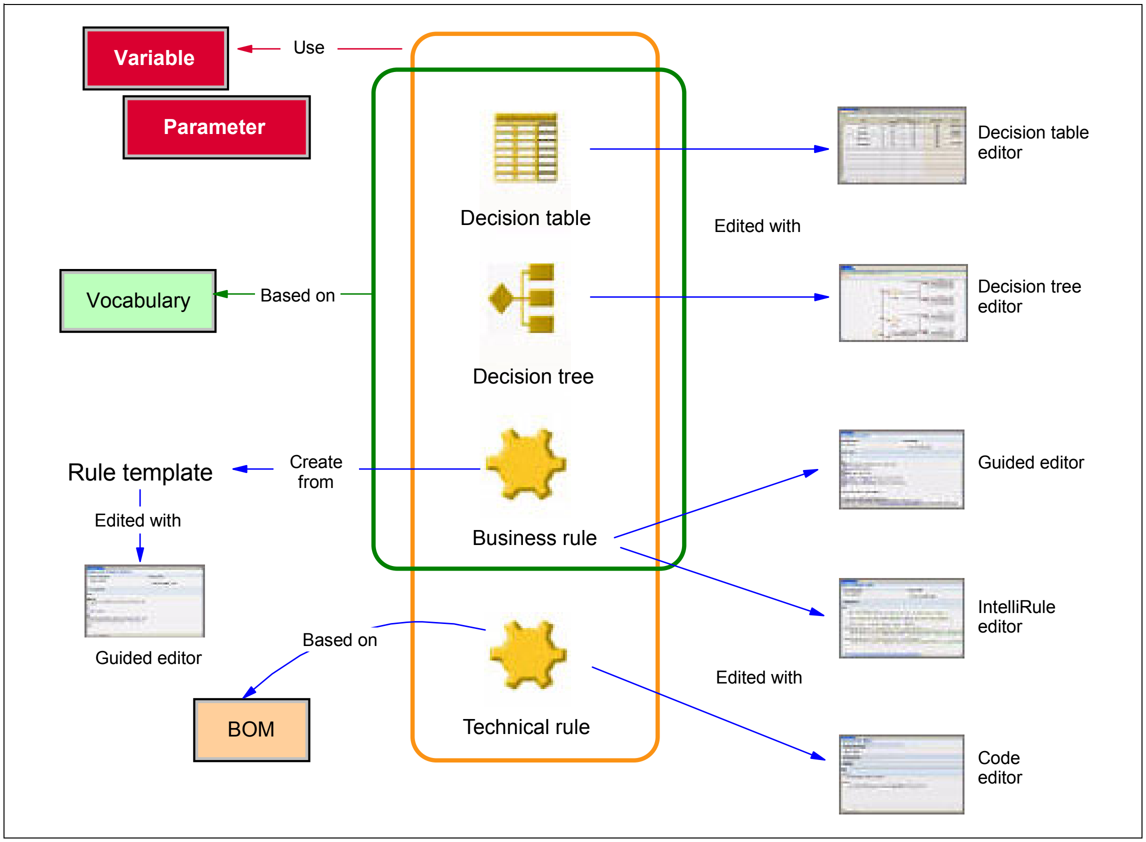


Figure 10-22 The relationship between rule artifacts and the editors that work on them

10.4 Business rule mining

Although, of course, you can build business rules in ILOG BRMS either by starting from a COBOL copybook or by manually developing the rules, imagine how much easier it is to extract business rule logic directly from existing COBOL programs. IBM Rational Asset Analyzer offers this function.

Rational Asset Analyzer, as described in Chapter 5, “Rational Asset Analyzer” on page 199, builds a repository of information about IT assets and supplies a mechanism for updating its repository and a user interface for viewing the information in meaningful ways. Rational Asset Analyzer support for rule mining is implemented as a natural extension to the assets that it already analyzes.

Rational Asset Analyzer scans a wide variety of files to derive a clear picture of business assets. Users can specify vocabulary (specific nouns) to guide Rational Asset Analyzer in focusing on the candidate business rules; you can also import the vocabulary from the ILOG BRMS repository. Rational Asset Analyzer manages these business assets as it does all of its other asset types. Rational Asset Analyzer provides counts and information to help assess and monitor the progress of rule mining projects.

Rational Asset Analyzer uses the word *business element* to refer to a term, a property, or both. It uses the word *vocabulary* to represent a collection of related business elements. The vocabulary collection can be scoped by an application asset or a business process asset, or it can be unscoped, in which case, it contains all business elements. A *business term dictionary* represents an alphabetical rendering of a vocabulary.

Rational Asset Analyzer uses the following asset types for rule mining:

- Business activity (activity): A named, structured process or task that produces a specific service or product for a particular customer or customers of a business. A business activity can be a collection of related business activities.
- Business category (category): An identifier assigned to business rules, business terms, and business term properties to let you filter which business terms and business term properties can be used when authoring a business rule.
- Business process model (process model): A named collector of business activities used to represent a core aspect of a business.
- Business rule (rule): A named statement, or set of statements, that defines or constrains an aspect of a business. Rational Asset Analyzer can capture business rules in either an unstructured form, a structured form, or both.
- Business rule mining (BRM) project: An optional, named collector of assets used to manage a given rule mining effort.
- Business term (term): A noun representing a concept that is used in the business.
- Business term property (property): A noun, of a specific type, representing an attribute or abstract quality that is associated with a business term. The relationships between business terms and business term properties are most often stated with the verb `has` (for example, car has driver) or the preposition `of` (for example, driver of car).

After the business rule mining data is in the Rational Asset Analyzer repository, you can use it to develop the rules using the ILOG rule editors, which are incorporated into the Rational Asset Analyzer product. You can export the rules from Rational Asset Analyzer into ILOG BRMS, either as structured or unstructured rules. After the rules are in ILOG BRMS, they are positioned for ongoing maintenance throughout the rule life cycle.

The synergy between Rational Asset Analyzer and ILOG BRMS is based on the common vocabulary between the products and the common rule mapping capabilities. The relationship between the two products is bidirectional and extends the value of both products.

Extracting business rules from a complex application can be a daunting undertaking. Rational Asset Analyzer allows users to build the repository of business rules in stages by using the concept of a project, allowing users to define the scope of the effort to be performed in a given iteration. Within a BRM project, you can limit the scope of the assets that Rational Asset Analyzer searches for (for example, a specific data element or concept) and build the business rules for that concept. This method allows you to incrementally build business rules one component at a time to meet the needs of your business community. Rational Asset Analyzer will also track the progress of BRM projects to give users a snapshot of how much has been done (and how much remains) in extracting rules for each asset.

Although Rational Asset Analyzer can play an important role in developing business rules, using it in combination with ILOG BRMS provides additional functionality. ILOG BRMS features, such as version control and the rule engine, are essential to providing a full BRMS life cycle. Using the business rule mining features of Rational Asset Analyzer jump-starts the process of identifying and automating business rules. The combination of Rational Asset Analyzer and ILOG BRMS results in an improved process for identifying and deploying business rules from start to finish.

10.5 Governance

With the ILOG BRMS, the need for governance is even more critical, because rules can be written and changed by people from various groups (for example, business users and IT development staff) and can also originate in separate tools (Rule Studio, Rule Team Server, and Rule Solutions for Office, although Rule Team Server is considered the true source). Adding Rules for COBOL introduces another layer of complexity, because the application that runs using the compiled source that was generated by Rules for COBOL is completely unaware that the COBOL source was generated by the BRMS product.

Every organization must develop its own governance practices that are suited to the policies, procedures, and organizational structure of the business. In this section, we present guidelines when developing the governance policies.

10.5.1 The goals of governance

BRMS governance includes the following objectives:

- Manage the life cycle of decision logic, from initial development through deployment and maintenance.
- Provide an organizational framework that instills confidence in all stakeholders.
- Ensure that business and development teams collaborate effectively. Development teams must hand over control of business rules to business users, and business users must accept responsibility for their actions in rules maintenance.
- Ensure that the project outcome meets expectations.

10.5.2 Components of a rule governance process

A rule governance process includes the following components:

- A purpose that is clearly stated in a charter and with goals
- A definition of stakeholders with their roles and responsibilities
- A process and a set of activities
- An assignment of the roles
- An entity to manage (or govern) the process
- A demonstration that the process is consistently executed

10.5.3 Implementing governance

Both business and development teams must actively participate in designing the governance processes. The governance process must be reviewed and adapted for each BRMS project.

10.5.4 Areas to address

You must complete these tasks:

- Develop the organization map to define roles for business rule management.
- Assign ownership and access control responsibility for business rules. Define who can create, read, update, and delete (CRUD) rules.
- Define the rule life cycle to determine a status for each phase of rule development (such as “validated” or “deployed”), and define who can promote a rule from one status to another.
- Define target deployment platforms, and who can deploy to which environment.
- Define each rule governance process:
  - Rule change management process
  - Rule authoring process
  - Rule testing process
  - Rule deployment process
  - Rule retirement process

10.5.5 ILOG BRMS support for governance

ILOG BRMS provides support for effective governance, including version management, access control, and reporting.

In Rule Team Server:

- Rules are managed rules in the Rule Team Server repository.
- Access is controlled through permissions according to user roles.
- Version tracking of rules and rulesets and baseline management are available.

In Rule Studio:

- Rules are managed in a file system through source code control (SCC).
- Synchronization tools are available to keep copies aligned across business and development environments.

Analyzing and reporting:

- Project-wide querying, analysis, and reporting features in both Rule Studio and Rule Team Server facilitate information gathering about rule status, rule access, and deployed rule versions.

10.6 The benefits of using WebSphere ILOG BRMS

You might wonder about adopting a product to support business rules, specifically about the ILOG BRMS family of products. What is in it for you? Actually, many benefits of ILOG BRMS exist for you as a z/OS developer at any stage in your career:

- You enhance your resume with cutting-edge technology.
- You make many aspects of your current work far easier and simpler:
  - Because you use a windowed user interface, which makes substantially better use of the window space, allowing you to see, edit, and work with multiple windows and views at the same time. For example, you can view your data model while you create your rule flow.
  - And, advanced application tools are available in ILOG BRMS can help you:
    - Seriously minimize the amount of typing you have to do, everywhere in the tools
    - Simplify navigation and scrolling work
    - Provide advanced test data manipulation facilities, vastly simplifying the work to build out your test suites
    - Provide Content Assistance so that you can point and click to choose valid data names, comparators, and actions from a pick list
- You build a better relationship with your business users by learning to speak to them in a vocabulary that is familiar to them.
- Although the end result output is familiar COBOL code, you build your Java skills as you write and change the rules.
- You do not have to rewrite or massively modify an application to take advantage of ILOG BRMS. You can upgrade one chunk of business rule logic at a time and incrementally modernize your application.
- The changes that you make to your application are likely to work better after you implement them, because you can vet them through rule analysis before you implement them.
- After the rules are written and an appropriate governance policy is in place, you can hand over trivial changes to your business users, leaving you time to work on more substantive projects.

Part 4

z/OS application maintenance and support

The complexity of production software, especially existing code, is well known. But, historically (and unfortunately), maintenance and support are often seen as less-than-prestigious assignments that are given to those individuals who are not technically up to developing new code. In this part of the book, we take a closer look at the various job roles involved.

Maintenance and support: The unsung heroes

In many z/OS shops, the jobs break down something like this:

- Executives and management positions
- Technical roles, which further break down into:
  - Systems, including database administrators
  - Operations
  - Business applications developers, which then break down into:
    - Development: Brand new application logic
    - Maintenance: Keeping it all working
    - Production support: Often combined with maintenance roles

Depending on the size of your staff and your shop's approach to the foregoing roles, individuals might wear different hats—but their tasks are well-known and differentiated.

It seemed to us, before we wrote this book, that there is not now—and never has been—any shortage of media material directed at and devoted to influencing the hearts and minds of executives and management. In fact, we have already discussed this subject in the first two chapters of this book.

Systems and operations personnel are typically well trained and/or they are too busy to read much, apart from ABEND listings, emails about job restarts, and the occasional article on database and system tuning. In our estimation, the systems and operations guys are all set. So that brings us to business application developers, and two final categories: writing new code and maintaining old code.

There has never been any shortage of materials about how best to develop new code. Every vendor under the sun, MI,S and computer science academics (it seems like almost anyone with a PHD) has opinions aplenty. And now they have a blog or electronic media column in some web journal; perhaps even an entire magazine devoted to how best to "write new stuff." So, for those of you doing development—we think you are all set. This leaves maintenance and support personnel, whom we address as the *unsung heroes* in an I/T shop. Why is that?

Many (far, far too many in our opinion) look at software maintenance and support in the same way that an airport or hotel looks at maintaining the cleanliness of its floors, terminals, and rooms. *"Yes, we have to do maintenance. Yes, it matters (we guess). But it is obviously not as important as, say, constructing new airports or new hotels, or flying planes."*

However, while that attitude works (to a point) in physical plant maintenance, it is *not* the right attitude (or analogy) for software maintenance.

The right analogy is actually doing maintenance on the planes themselves, and doing “production support” on broken planes. If done poorly, bad maintenance and support leads to delays, angry customers, loss of business revenue, and worse. But when done well, quality maintenance and support leads to “satisfied everyone”—contented staff, happy customers, relaxed and upbeat pilots, and so on. The only unsatisfied folks are your competition. The problem that we are trying to address though, is—who is applauding the maintenance and support teams of the world, and helping them do their jobs better?

Just look around at the media and media events (books, articles, research, seminars, conferences, vendor presentations, and so on) produced over the past 30 years. What percentage of it addresses the ongoing needs of “better maintenance and support.” You might be shocked to find out that it is almost zero. Since the middle 1980’s, there has been almost nothing said or written or researched in this area.

That is most unfortunate, because (we have to ask this rhetorical question) what is more important to the health of a business application division than:

- Maintenance that is effective and efficient
- Production support that works (fixing things completely, correctly, and in a timely manner), and that is also effective and efficient

At IBM we have tried to answer these concerns by writing and publishing this book. What is more important than ensuring that the applications running transactional operations at your company are well cared for, and that the individuals doing that work are appreciated and given their rightful attention and respect?

And specifically, we hope to answer this question in the next section, which is about using better processes and vastly better tools to improve the maintenance and support world in your shop.

Furthermore, we hope to supply the reader with an additional asset for building their professionalism.

A quick look at, and definition of, maintenance

Or perhaps not such a quick definition—In our work with z/OS shops in the Global 2000, it has seemed that every shop is almost completely distinct in its:

- Development life cycle (including maintenance and support)
- Approach to maintenance staffing and projects
- Development tools
- Development and testing environment

Entire books might easily be devoted to the maintenance process, the “righteous methodologies” and best practices/lessons learned. This almost certainly might open up a Pandora’s Box of controversy over “*the one true and best way*” to do maintenance. And much as we might be tempted to jump into the ring with practitioners, mavens, and gurus, we will resist the urge and focus on three simple definitions: production support, trivial maintenance, and non-trivial maintenance.

In this chapter we have three sections on maintenance. They are actually detailed use cases:

- Production support
- Trivial maintenance
- Non-trivial maintenance

The authors of these sections do excellent jobs of delineating the details and differences among these. But first, we summarize them here as follows.

A quick look at, and definition of, production support

Production support is basically all of the tasks that surround fixing production software ABENDs, and anomalies. It is typically, though not always, composed of two distinct phases:

1. Immediate software triage: Find the problem in the application that caused the ABEND condition. Fix, or if necessary, bypass the problem, to get the application back into production quickly, especially as the business often requires it. An example might be a module in a batch stream that must complete in order to produce files needed by downstream jobs, and so on. Immediate triage implies determining “what happened” and how to fix the code so that the business process can complete. This is done with the understanding that “how or why” something happened is a luxury that the time-dependent nature of production support often does not allow. Immediate triage often takes place off-hours (although not the case for online systems).
2. Root cause analysis and resolution: Find the “how or why” for the ABEND. In this phase, which usually occurs during the standard work day, production support staff, or maintenance staff performing the job of production support, analyze the application code. They are looking to uncover the precise reason for the application failure in order to fix the problem and to understand the implications on other modules and even other applications. Finally, the team will fix the code (ostensibly, permanently), test it, and promote the code back up into production.

Phases

The reason we say that the foregoing phases are not always how production support is done is that occasionally—and we have to admit, in our experience, rarely—when the production support staff pinpoints the software problem or coding issue, the resolution is apparent, and both triage and analysis/resolution can be done in one step.

Also, for some applications (especially mission-critical/online transaction processing systems in IMS or CICS), the need to find AND fix is so critical and time-dependent that you have to both—and do it quickly. Using the equation for Risk = "Probability times Consequences," then the risk of more financial losses due to software ABENDs predicates that certain production support issues are of the "Find it and Fix it NOW!" variety. (This is because the cost of allowing application down-time is just too expensive. In fact, we have heard that in certain financial systems, down-time costs for online systems can run into the millions of dollars per hour).

Op ed

And yet, even while writing this and thinking through the ramifications of less-than-stellar maintenance practices (far and away the primary reason for production support problems, and the costs to your organization that stem from them)—it literally baffles us that in the world of software journals, writings, conferences, and so on. more space has not been devoted to this topic. It is almost as if the world is a gigantic junior high school, and the "cool kids" who rule the social hierarchy are focused on what is "new and shiny"—not what matters.

Trivial maintenance

Trivial maintenance is any software change to an application that is alleged to be simple, confined, and of short enough scope to be accomplished somewhere between an hour and a day. Such requests are often referred to as "one-liner changes"—and of course—they seldom are.

Somehow it seems in the nature of software project managers and architects to underestimate almost everything work and task-related that deals with production application code. We cannot explain why this is the case, because most of these folks matriculated from a production software and project world. And yet, when they become managers and analysts, they transform from grim realists (those of us actually responsible to *do* the technical work), to naive optimists (those that *assign* the work).

- "Hey Jim. Can you make this simple fix to module XYZ? It ought to be a one-line change, right?"

Well, back in 1981, it might have been simple. And perhaps if the application had just gone into production, was well structured, well documented, and well coded, *and* if the programmer had a vast experience and first-hand application knowledge, a "simple" fix and one-line change might exist.

But it is actually 2011 (or after) as you read this. Production applications have rarely been well maintained with updated comments, code restructuring, consistency, and so on. In our experience, what has happened to the z/OS applications that transact business in most shops is that they have been *patched*, meaning that the code has been modified, with statements added, changed, and deleted without regard to the systemic integrity of the design or original module.

And over the years or decades, the net result of dozens to hundreds of patches is complex, unstructured code (with GO TO EXIT statements), and where the documentation means less than nothing (it only serves to obscure the working logic in the source). One-liner changes, like the myth of a balanced federal budget, are nothing more than spin.

So, we have named our use case "trivial maintenance"—which actually (and unfortunately) *trivializes* the work involved—although at least the term is not the fictitious depiction of a "one-line change".

Still, trivial maintenance "weighs less"—it is simply smaller in code volume and effort than non-trivial maintenance projects.

Non-trivial maintenance

Non-trivial maintenance is any software modification that will take over a day—and probably a lot longer. We wish we had a more precise and singular definition of this kind of work, but it comes in many sizes and shapes, from sweeping changes due to large-grain business enhancements, to adding new functionality, re-engineering, re-writes, optimization, changes due-to regulations, and so on.

"Non-trivial maintenance" implies that this (task) might be difficult, challenging, time-consuming, unwieldy and large—possibly crossing application boundaries.

Non-trivial maintenance requires structured, formal project management, with controls, standards, and rigor; and ought to be staffed with senior, experienced software professionals armed with quality tools (like the ones covered so far in this book, for instance) and provided with a systematic proven methodology.

Summary

Both production support and maintenance require a tremendous amount of software experience *and* skill (the two of course, are not synonymous).

Both require deep competencies in z/OS technology, software analysis, programming, and design. And especially—this is almost never mentioned in a world ruled by “better, cheaper, faster” (euphemism for “cheaper, cheaper, cheaper”)—both production support and maintenance require a deep understanding of the underlying business/semantic domain (what exactly *does* this code do, and how does it contribute to the financial bottom line of your company).

It seems to us that few jobs in the software world are more important—or more difficult to do well—than production support and maintenance.

So, if production support and maintenance are what you do, we salute you—the unsung hero in your organization. We hope that these Use Cases will help, and provide insight into how the products of the optimized life cycle can help you do your job better and faster, with less stress, and ultimately, with money savings to your organization.

This might not be a bad deal—and we hope it is a deal you get to make—by using the products described herein and allowing them to improve your world.

11

Production support use case

The term *production support* in this context refers to the work that is done by the applications staff to support the successful completion, typically overnight, of production batch processing, which forms a critical part of an application. In many sites, the staff must complete this batch work by a certain cutoff time to allow the online system to be fully functional and ready for another day of activity.

Even though companies might move toward continuous processing and away from the online/batch split, batch jobs that need to be fixed quickly will always exist. Developers who support these important batch processes provide a vital service to the organizations that employ them. In this chapter, we show how the tools and processes of the optimized life cycle can assist production support developers to address the issues they face.

11.1 Production support process flow

People often compare the production support role in an IT shop to the triage principles that are employed in the emergency room of a hospital. Even though production support is not a matter of life and death, the analogy fits when considering the prioritization of the work to be done. The issues need to be brought to the attention of those people who are best equipped to deal with them. Tasks must be assessed and prioritized based on their urgency. Jobs on the critical path of the batch stream need to be fixed before jobs that are not.

Also, it is important to capture information at the time of the ABEND. This information will be valuable in both handling the immediate issue and also later when investigating the root cause of the problem. After the immediate concerns are dealt with effectively, you can identify the root cause and develop, test, and implement its solution (Figure 11-1).

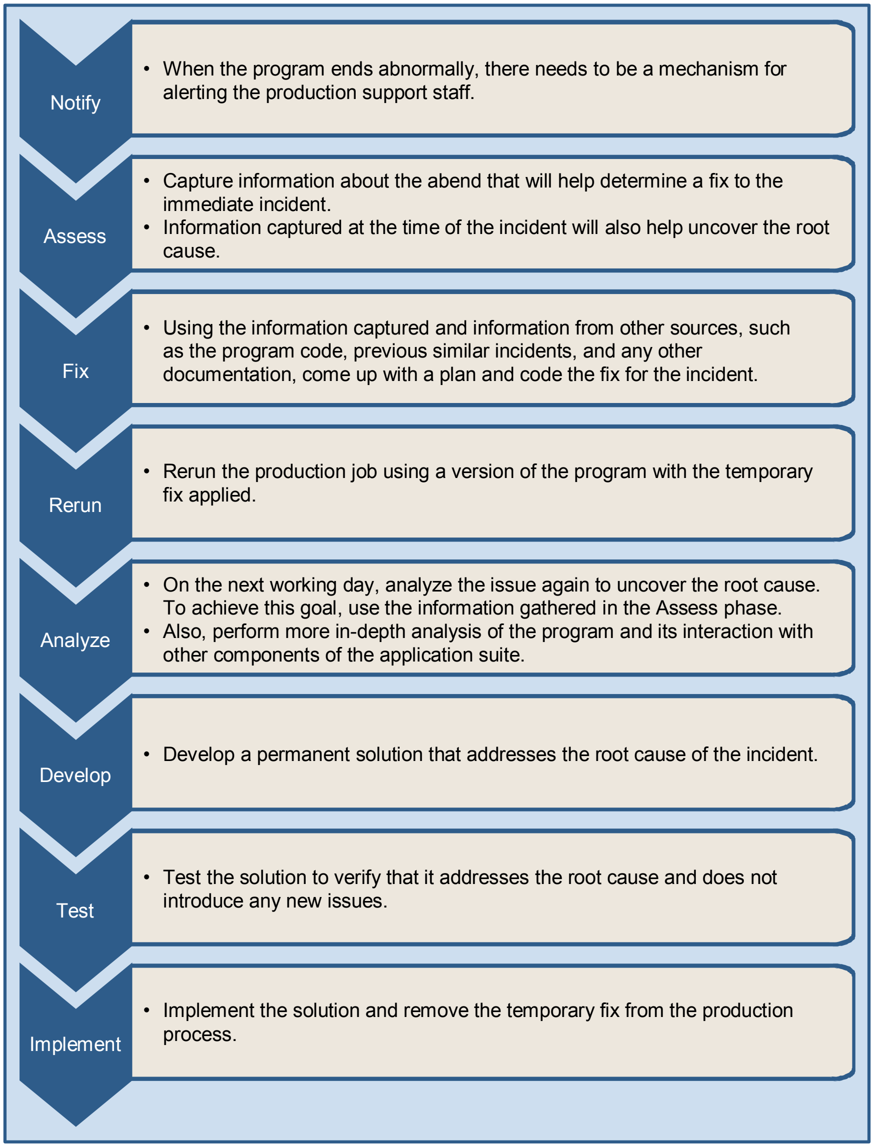


Figure 11-1 Production support process flow

11.2 The scenario

In this case, a batch job has ABENDed and a call has been placed to the on-call production support applications programmer. The Notify phase is the first of the eight phases. Notification can be as sophisticated as automation software sending a Short Message Service (SMS) text to on-call staff, alerting them to a failure. Or, it might be as simple as a computer operator on a shift reacting to a non-scrolling message on the system console, issued by the failing job. It is extremely important that notification happens quickly to allow the maximum time for the next three steps.

11.3 Assessing, fixing, and rerunning

The programmer is faced with S0CB ABEND in subprogram CLCLBCST of TRMTSRCH. The job log shows that Fault Analyzer has captured the ABEND information. Fault Analyzer lists the following useful information in the job log (Figure 11-2):

- Program, module, and source line number
- ABEND information, including a short description (`Decimal-Divide Exception`).
- The fault ID and the history file where the ABEND information can be found

> **Configuration:** To gain the maximum information from the tools, ensure that you configure the programs with the suggested options. Refer to the production documentation for these settings.

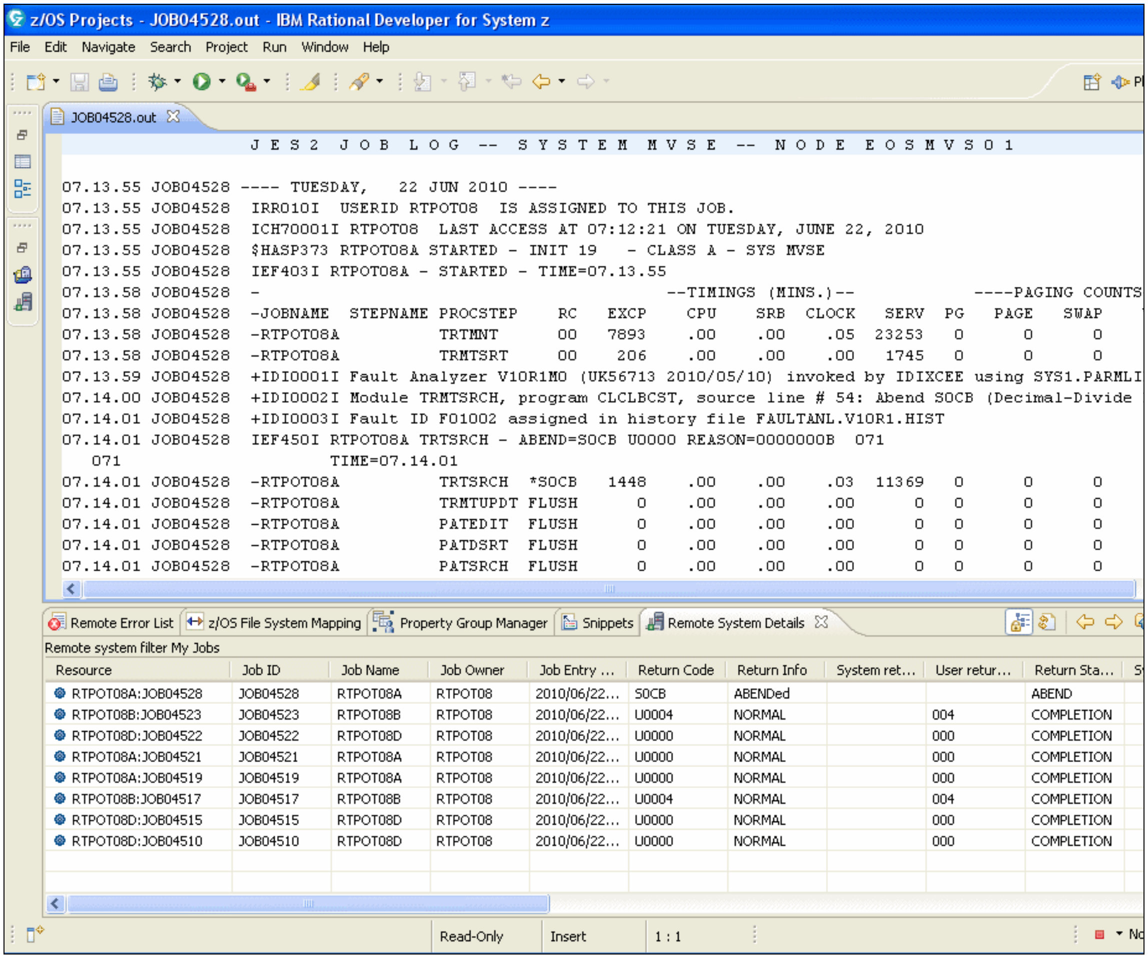


Figure 11-2 Job log showing ABEND

To view the Fault Analyzer report, the programmer switches to the Fault Analyzer perspective. The history file displayed in the job log is added through the context menu (Figure 11-3).

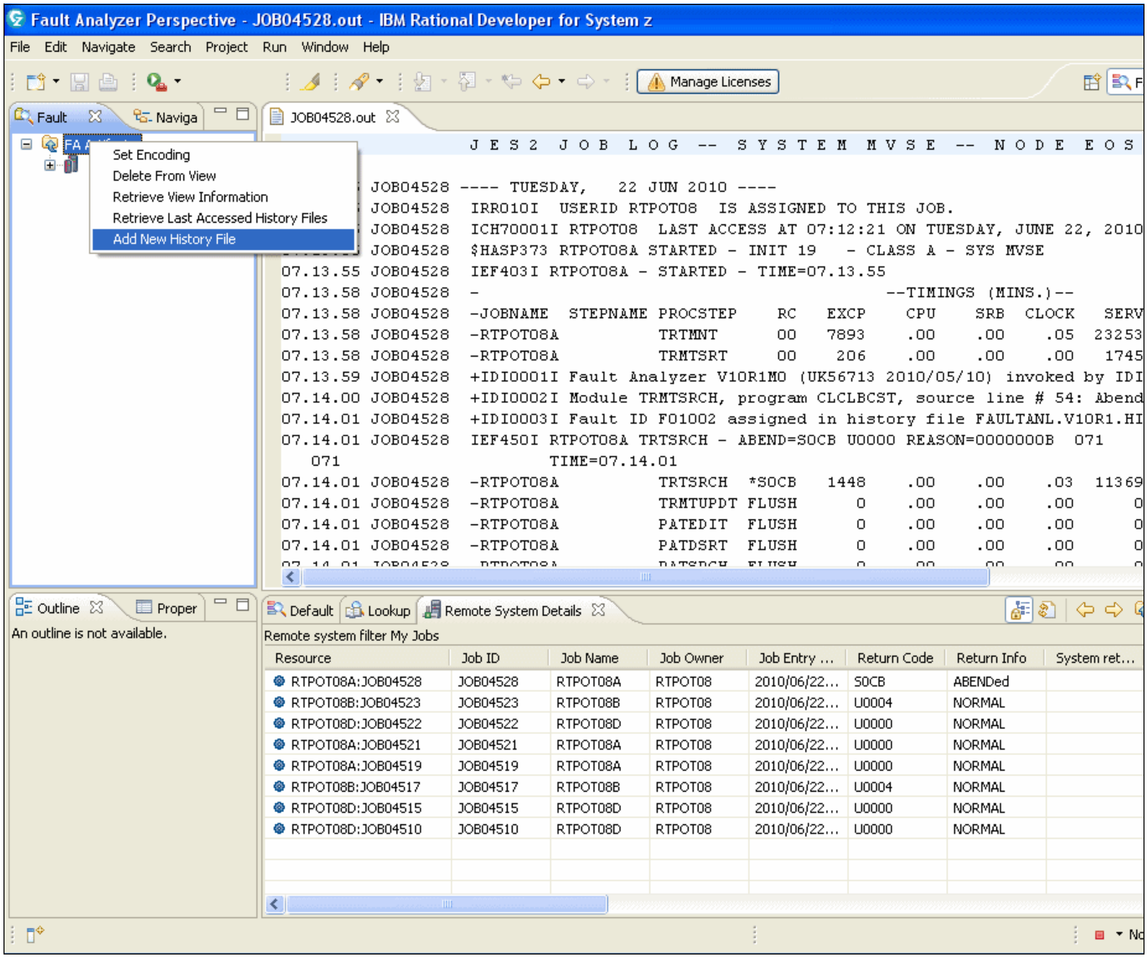


Figure 11-3 Adding a new history file

The Fault Analyzer Report (Figure 11-4) shows the source line that immediately preceded the failure. In this case, the divisor in the Compute statement on line 54, `PATIENT-COPAY`, has a value of 0. Hovering over PATIENT-COPAY in Rational Developer shows the programmer that the field is part of the linkage section of the program. The field is passed to the program from the calling program, which, in this case, is TRMTSRCH.

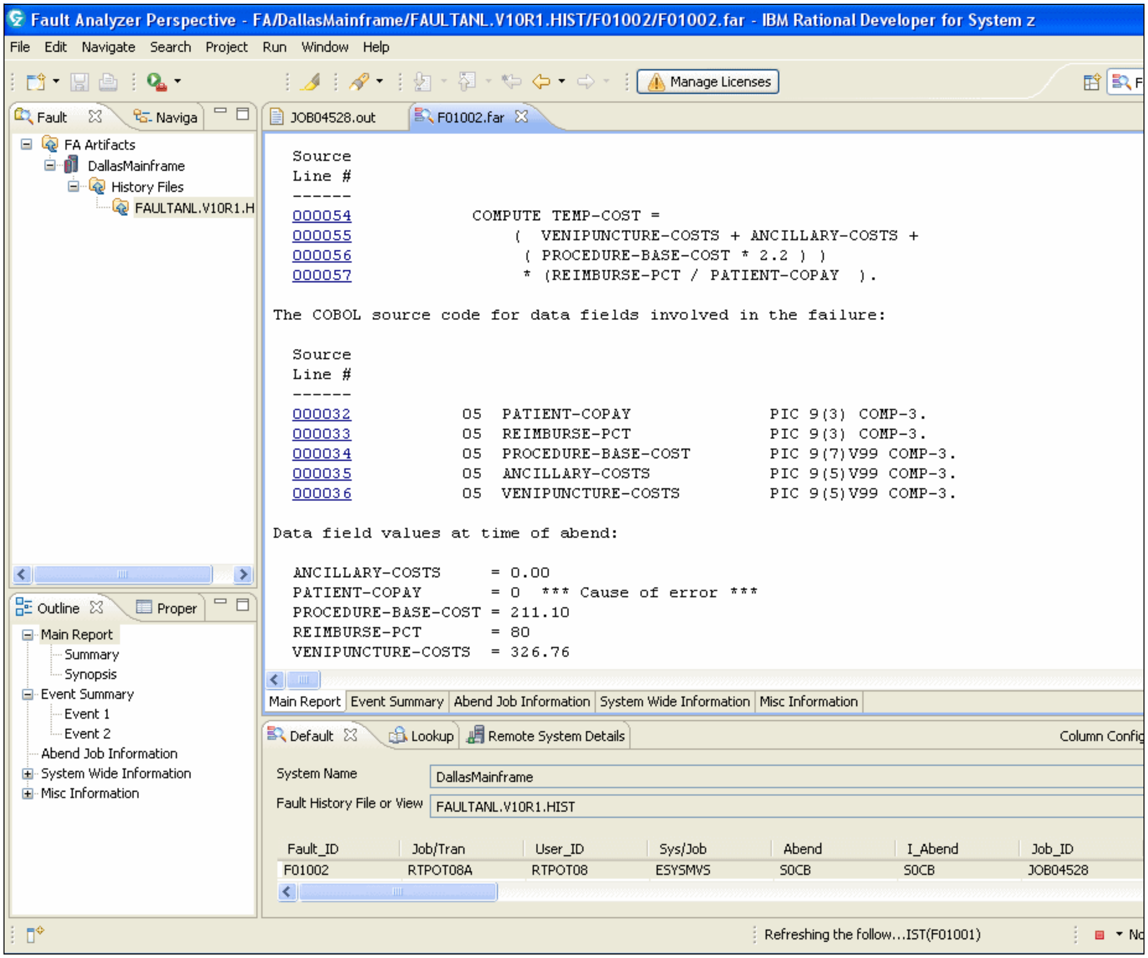


Figure 11-4 Fault Analyzer Report showing the source line that immediately preceded the failure

The programmer inserts code into the program to check if the field is 0 and to move a valid value to it. The programmer consults the application documentation to discover that the default co-payment factor is 100, but that certain states will have a lower factor set. The programmer decides to add a test to test for the field being zero, and if so, to move 100 to the field to eliminate the S0CB.

The programmer adds the code and recompiles the program to a temporary library. The programmer reruns the production job successfully from the temporary library, which allows downstream processing to complete within the window allowed for batch processing under the service level agreement (SLA) in place (Example 11-1).

Example 11-1 Adding emergency fix code to the program

```
100-CALC-LAB-COSTS.
**  Emergency Fix to allow batch to complete
     IF PATIENT-COPAY = ZERO
            MOVE +100 TO PATIENT-COPAY.
     COMPUTE TEMP-COST =
          (  VENIPUNCTURE-COSTS + ANCILLARY-COSTS +
           ( PROCEDURE-BASE-COST * 2.2 ) )
           * (REIMBURSE-PCT / PATIENT-COPAY  ).
```

11.4 Analyzing the root cause and developing a solution

At the next opportunity, an analyst within the applications area supporting the batch job needs to determine the root cause. This effort involves trying to understand how and what sets the content of the PATIENT-COPAY field within the patient master file. You can use Rational Asset Analyzer to determine where a field is modified within a set of application components, including programs and copybooks. Also, you can integrate Rational Asset Analyzer with Rational Developer so that you can perform the searches from a Rational Developer perspective:

1. Right-click the project that contains the elements to search.
2. Then, within the search window, click the **RAA Integration** tab (Figure 11-5). Enter the `PATIENT-COPAY` field name in the search string, select **Data Element** to specify to search for a data element, and select **Modifications** to limit the search to modifications of the data element. Click **Search**.

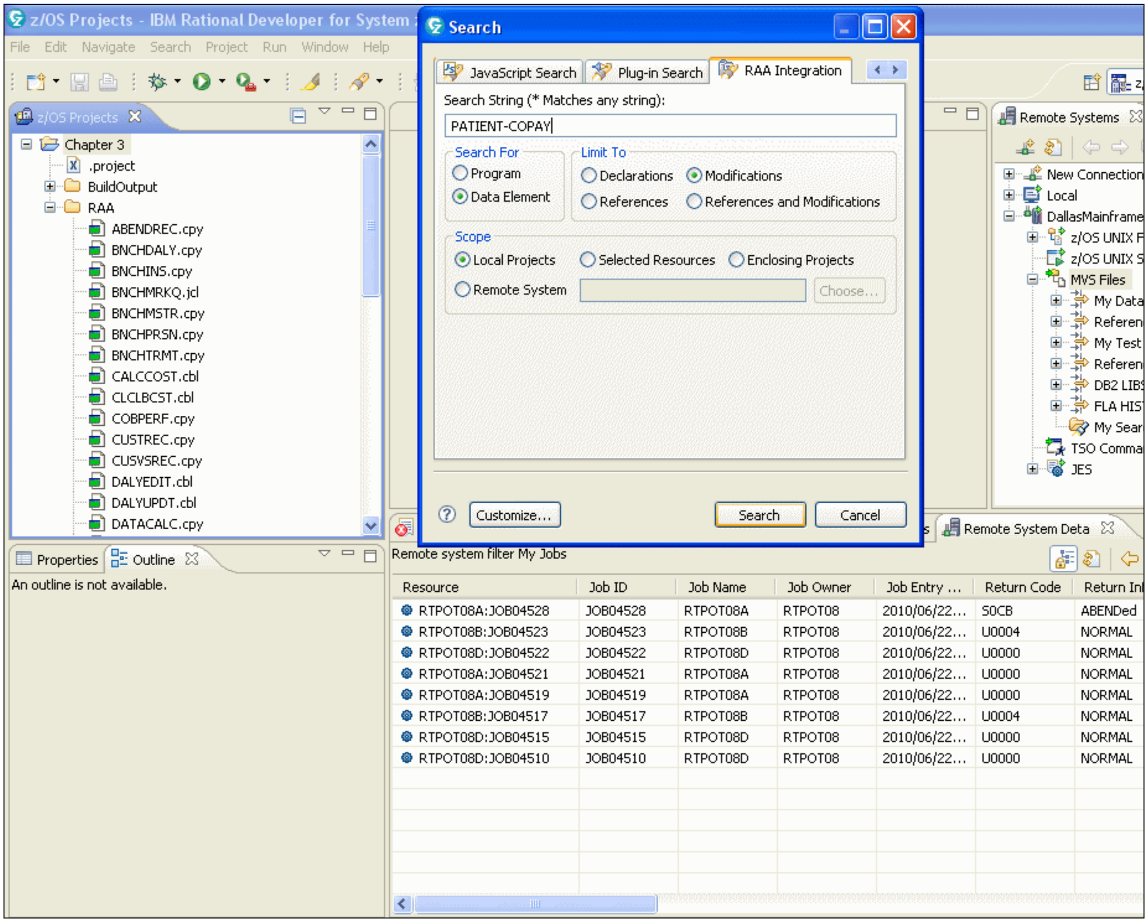


Figure 11-5 Integrating Rational Asset Analyzer with Rational Developer to perform searches

The search reveals a number of references. The analyst must investigate each reference. A modification within PATSRCH shows that another field called COPAY from the PATIENT-MASTER-REC (patient master file) is used to populate PATIENT-COPAY (Example 11-2).

Example 11-2 Investigating each reference

```
300-CALC-EQUIP-COSTS.
           MOVE "300-CALC-EQUIP-COSTS" TO PARA-NAME.
           MOVE PATIENT-ID IN INPATIENT-DAILY-REC TO
             PATIENT-KEY,
               PATIENT-ID IN CALC-COSTS-REC.
           READ PATMSTR INTO PATIENT-MASTER-REC.
           MOVE COPAY  TO PATIENT-COPAY.
```

The task now is to trace how COPAY in PATIENT-MASTER-REC is populated within the entire application. Again, the analyst uses the integration between Rational Developer and Rational Asset Analyzer (Rational Asset Analyzer integration) to complete an intelligent search of the application. In a similar manner to the previous search for PATIENT-COPAY, the analyst searches for modifications of COPAY. This search reveals that COPAY is set to STATE-FACTOR, which, in turn, is based on an evaluation of the EMP-STATE field (Figure 11-6 on page 490).

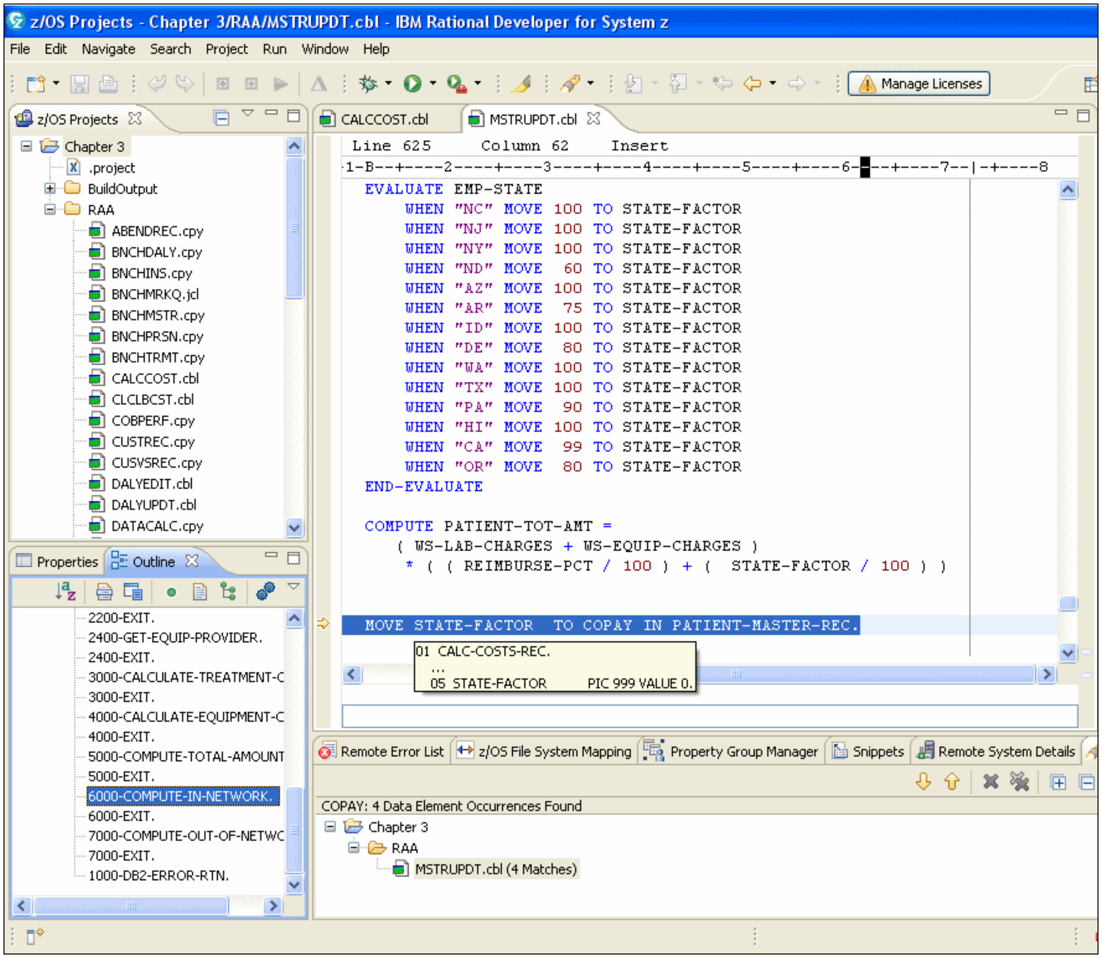


Figure 11-6 Searching for modifications of the COPAY field

The analyst notices two things from looking at the code excerpt and from hovering over the field:

- The list of state codes (EMP-STATE) is not exhaustive, and there is no WHEN OTHER clause on the EVALUATE to catch unexpected values.
- The initial value of STATE-FACTOR is set to 0 in the working storage.

If a state code (EMP-STATE) value other than those codes listed explicitly in the EVALUATE statement occurs, COPAY will be left at the initial value of 0 rather than the accepted default value of 100. To test the theory that there are values of EMP-STATE possible that are not catered for, the analyst uses Rational Developer and its Debug perspective. In the debug session, the analyst sets up the following test:

- Sets a breakpoint at the following statement:

  ```
  MOVE STATE-FACTOR  TO COPAY IN PATIENT-MASTER-REC
  ```

- Highlights the relevant fields (EMP-STATE and STATE-FACTOR), right-clicks them, and chooses to monitor them. In this way, the analyst can see their values change as the program is debugged.

We show the results after processing a couple of records in the debug session (Figure 11-7). Notice the values of EMP-STATE and STATE-FACTOR in the Monitor window.

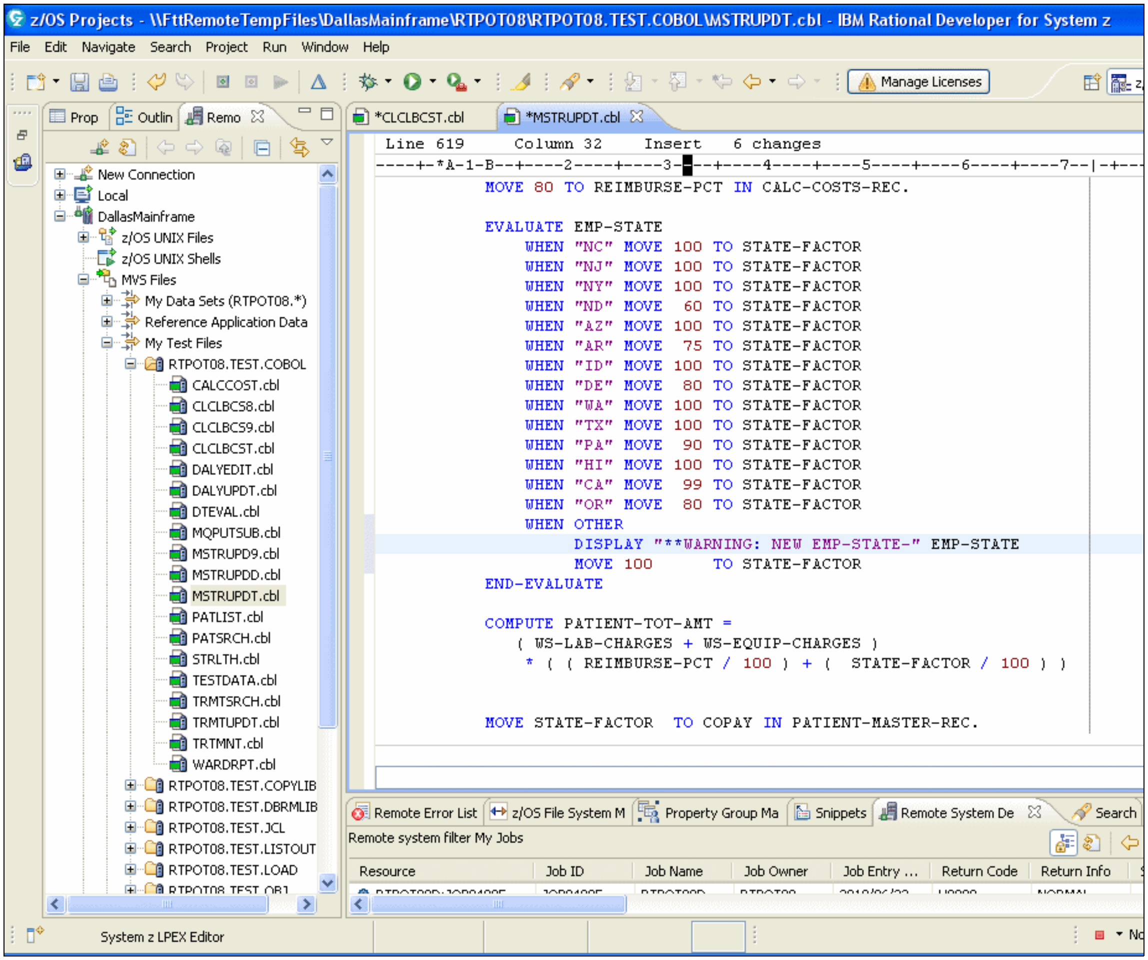


Figure 11-7 Showing the results after processing a couple of records in the debug session

EMP-STATE had a value of "IL". This value is not catered for in the EVALUATE statement, so at the breakpoint, just before COPAY in PATIENT-MASTER-REC is modified, STATE-FACTOR is still at its initial and invalid value of 0.

It is a good practice with EVALUATE statements to always include a WHEN OTHER clause. This practice makes the program more resilient. Normally, the addition of new states from a business perspective means that the application area will be notified so it can make the necessary program changes. However, this notification does not always happen. Programs need to be written to expect the unexpected.

The initial intention of the application was to have a default co-payment factor of 100 but to allow for lower factors for certain states. The applications team agreed to the change that is shown in Figure 11-8. The change sets the factor to the default value of 100 but also shows a warning message on the job log. It might be better if these exceptions are also written to an error file, but, for this example, we use a display to the job log only. The point is to highlight the issue, so the applications team can follow up the issue and address it rather than let it keep occurring and causing unforeseen issues as it did in this scenario.

The fix has three effects. First, the fix implements the original intention to have a default factor set. Second, it stops any future repetition of this S0CB ABEND. Third, the warning message highlights to the applications team when a new state is now incorporated into the database. This message provides an opportunity to discuss the introduction of the new state with a business representative and to ensure that the co-payment factor chosen for each state is appropriate rather than merely defaulting (Figure 11-8).

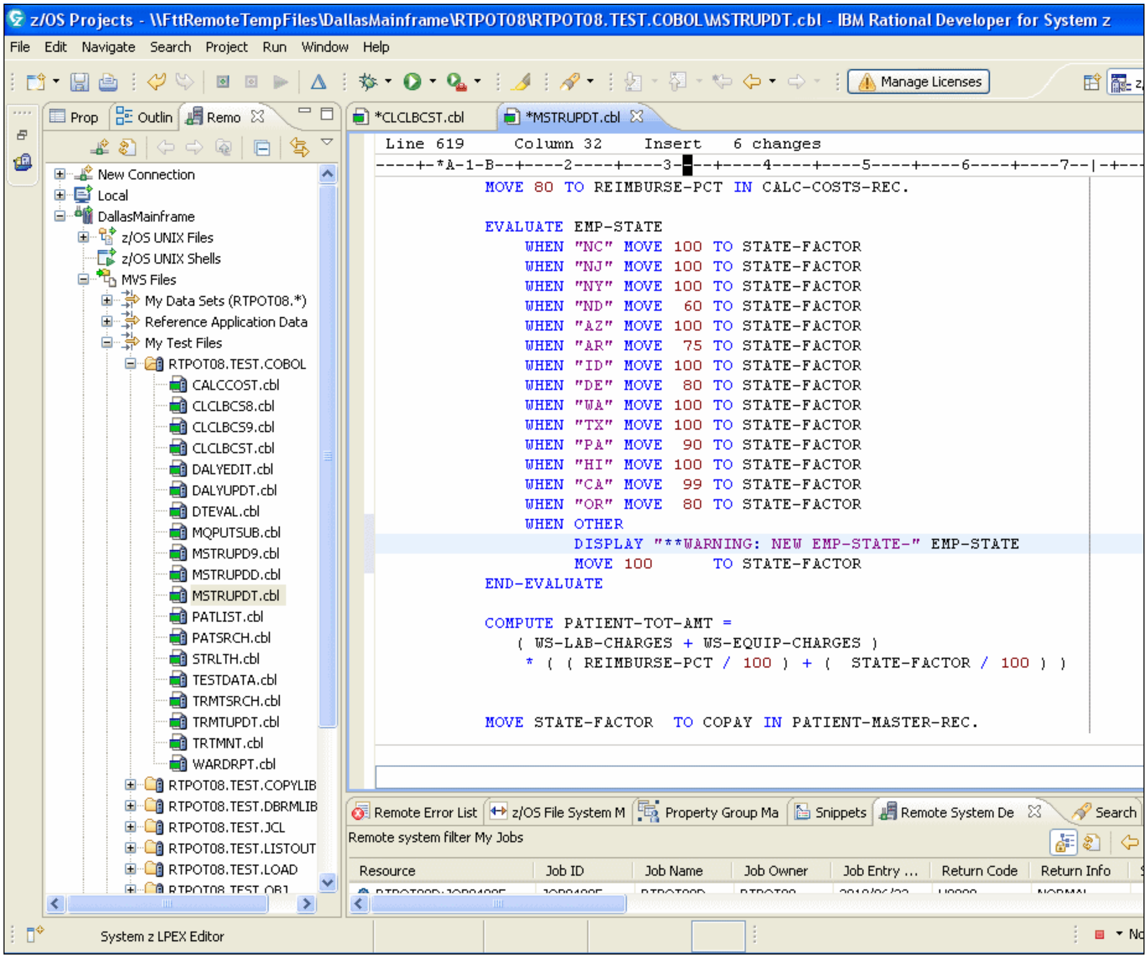


Figure 11-8 Determining the fix

11.5 Testing and implementing the solution

Implementing the solution involves following the development and testing methodology that is adopted by the site: traditional approaches such as the waterfall method; collaborative approaches, such as the Iterative Approach using RUP or Scrum; and Agile Methodology, which can enabled by the use of a collaboration tool, such as IBM Rational Team Concert for System z, as shown in Figure 11-9 and Figure 11-10. You can integrate Rational Team Concert for System z with Rational Developer, as we did with Rational Asset Analyzer, to form an important part of the IBM Enterprise Modernization Workbench.

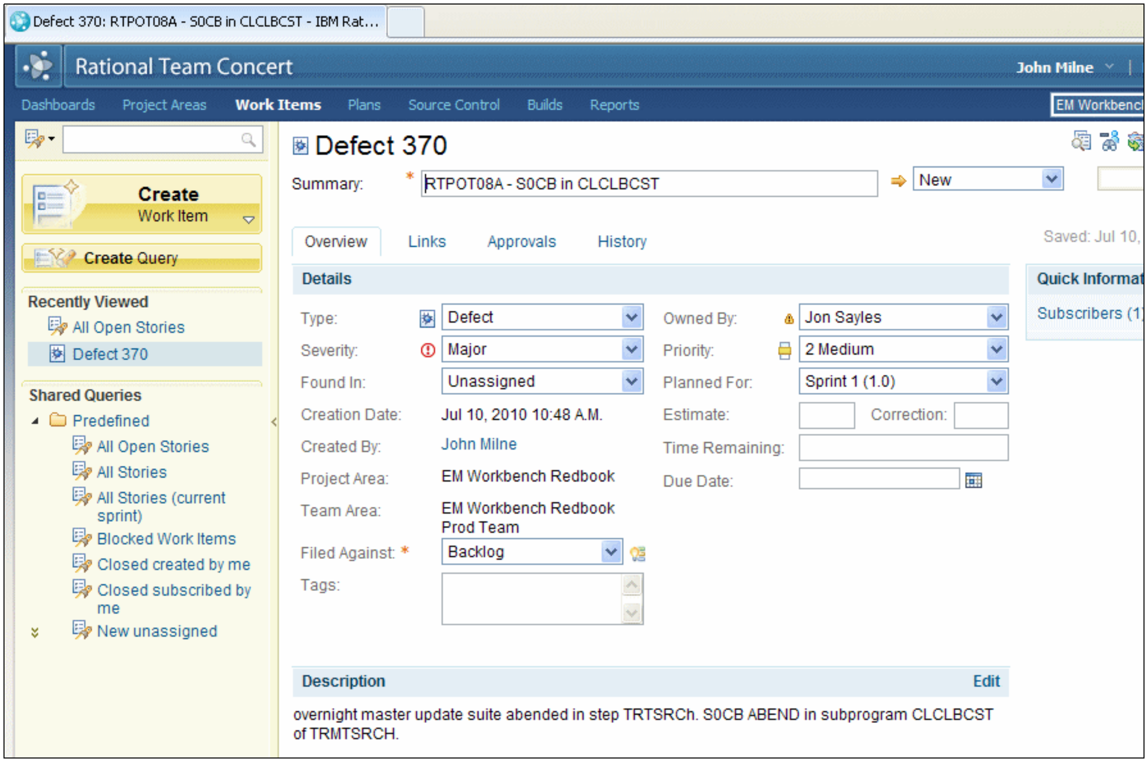


Figure 11-9 IBM Rational Team Concert for System z

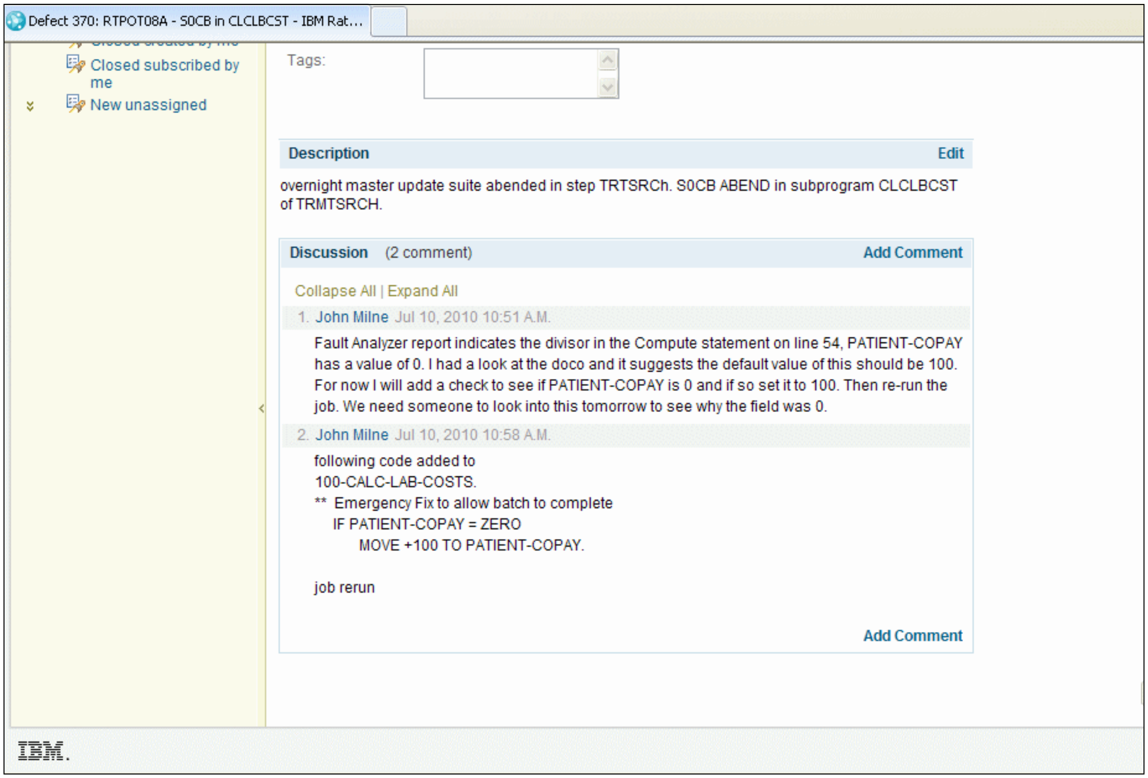


Figure 11-10 IBM Rational Team Concert for System z

It is important to test that the changes have the desired results. After retrieving the programs, the applications programmer makes the changes in a test environment. The applications programmer runs the job in that test environment using sample data or copies of production data. The test reveals that a number of states were introduced into the database without changes being made to MSTRUPDT to ensure that the COPAY field was set correctly (Figure 11-11).

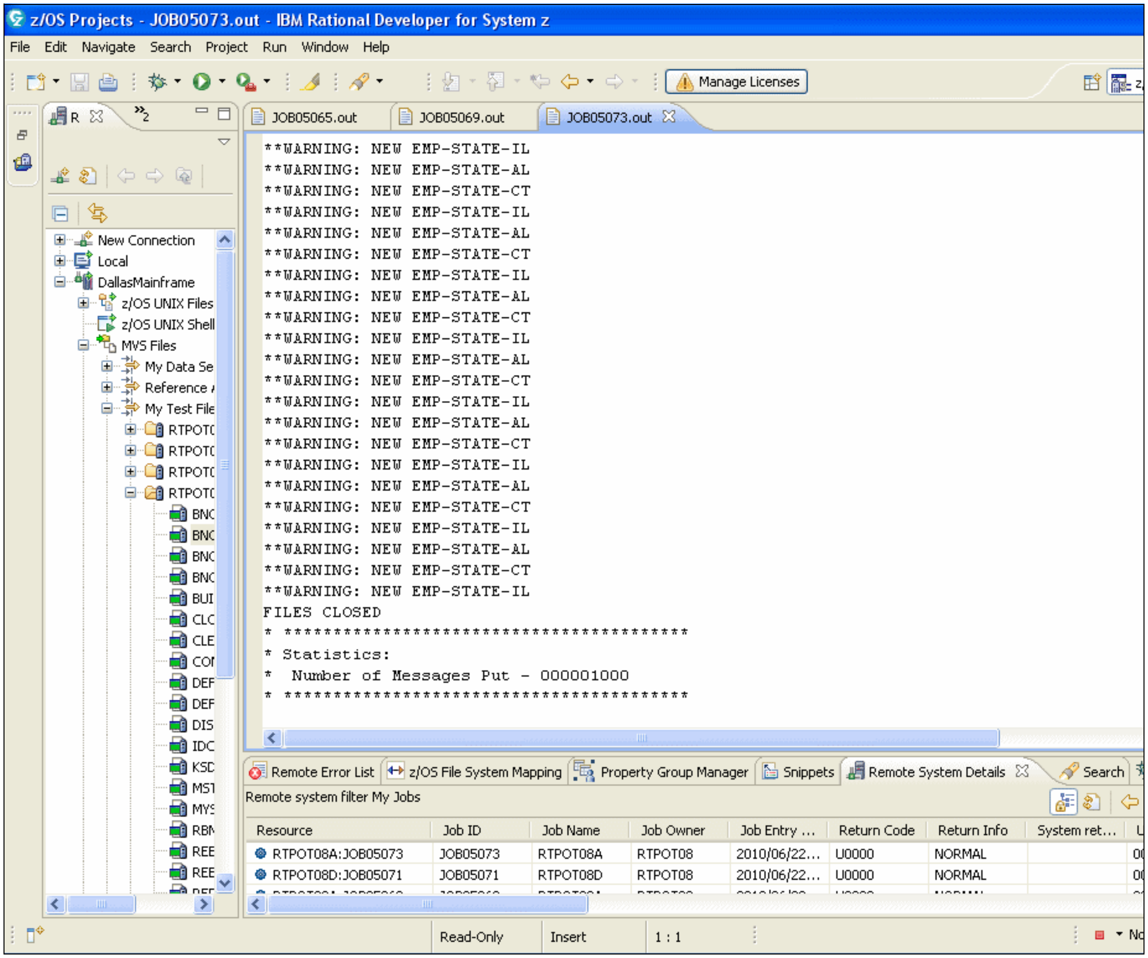


Figure 11-11 Results of test

More testing might be required before the program is migrated to production, including a system integration test of the whole system to ensure that this change does not have unforeseen consequences downstream. At the same time that the solution to the root cause is migrated to production, the application programmer must remove the temporary fix that was made to a version of CLCLBCST to ensure that the new, full solution is picked up when the suite next runs.

11.6 Summary

In this chapter, we have demonstrated how the tools of the optimized life cycle can be used to improve the response to a production support ABEND. It is possible to investigate these types of issues without the tools discussed and without ABEND analyzers, such as Fault Analyzer or third-party alternatives. COBOL can provide a symbolic dump, and if a compile listing with the appropriate options has been kept, you can derive the failing line of code from the standard module/offset ABEND information. Also, an applications staff with in-depth application knowledge and experience can quickly drill down on issues without much help. In fact, this method is how production support issues were investigated and might still be at certain sites.

We demonstrated in this chapter how the tools that are now available and the integration between them can benefit individuals who tackle these issues on a daily basis. These production support incidents are often stressful events that, if not addressed in a timely manner, can have major negative effects. The tools can help you reclaim time that in the past was lost in identifying the point of failure and identifying a fix. This time savings can be the difference between getting systems back on track quickly or not.

12

Trivial Change Request use case

In this chapter, we document the use of integrated tools in the optimized life cycle to process what we call a “Trivial Change Request”. Specifically, we focus on using the tools introduced in previous chapters to accomplish the various tasks associated with this kind of request:

- Specifying the business requirements and functional requirements
- Performing a static (and possibly dynamic) analysis of the impact that a proposed change will have on the application, as well as upstream and downstream processing
- Performing the development activities, including coding and testing
- Highlighting the benefits of using the tools over ISPF

12.1 Trivial Change Request use case introduction

A change request can be defined as a call for modifying the behavior of a system due to normal business changes or because there is a bug in the system. A change request needs to go through a software development cycle in which the impact of the considered solution and effort is analyzed and estimated for that change, and then it has to be approved for implementation before work on it can begin. In short, the goal of the process is to produce a more robust system that better supports the business activities of our customers.

Based on the complexity and the effort required to complete a change request, we can term them as *trivial* or *non-trivial*. It so happens that most of the time, a change request that we consider trivial, in the end, turns out to be more than trivial. Here we consider a change request that is a classic example of one of these scenarios. In this section we show how to use the tools and processes of the optimized life cycle to complete the change request successfully.

12.2 Business requirements

The healthcare business application OnDemandHealth has the logic to compute REIMBURSE-PCT (reimbursement percentage), hardcoded in the programs as literal-based calculations. This logic needs to be changed such that the value for REIMBURSE-PCT is calculated depending on the table values. Note that while most production applications will, of course, already be variable-based in their computations and algorithms, we consider this change a “category” of change request that appears to be “trivial” in scope.

12.3 Functional requirements

At this point, communication with the client is required to understand the business rule that will be applied to compute the value of REIMBURSE-PCT. After all the clarifications are received, the Functional Requirement document is prepared, which describes in brief the functional modification to the system. An applications programmer or systems analyst then documents a clear (and probably standardized) statement of modification – similar to this:

"The OnDemandHeath system will be modified to incorporate the calculation of the REIMBURSE-PCT based on the values of the DEDUCTIBLE, COPAYMENT, OOP MAX, COVERAGE LIMITS, and the NETWORK flags" (Example 12-1).

Example 12-1 On Demand health system

```
      IF DEDUCTIBLE > 0 AND
         COPAYMENT > 0  AND
         OOP_MAX < 1000  AND
         COVERAGE_LIMITS > 10000 AND
          IN_NETWORK_REQ = 'Y' AND :IN-OUT-NETWORK = 'Y'
          MOVE 80 TO REIMBURSE-PCT
      ELSE
      IF (DEDUCTIBLE > 0 AND COPAYMENT = 0) OR (DEDUCTIBLE = 0 AND
COPAYMENT > 0) AND
         OOP_MAX < 1000  AND
         COVERAGE_LIMITS > 10000
         COVERAGE_LIMITS > 10000 AND
          IN_NETWORK_REQ = 'Y' AND :IN-OUT-NETWORK = 'Y'
          MOVE 60 TO REIMBURSE-PCT
      ELSE
         IF IN_NETWORK_REQ = 'Y' AND :IN-OUT-NETWORK = 'N'
          MOVE 40 TO REIMBURSE-PCT
         ELSE
          MOVE 30 TO REIMBURSE-PCT.
```

The Project Manager or the Project Lead, at this point of time, will open a Work Item in RTCz documenting the approved Change Request and assign it to the resources required to complete the task (Figure 12-1).

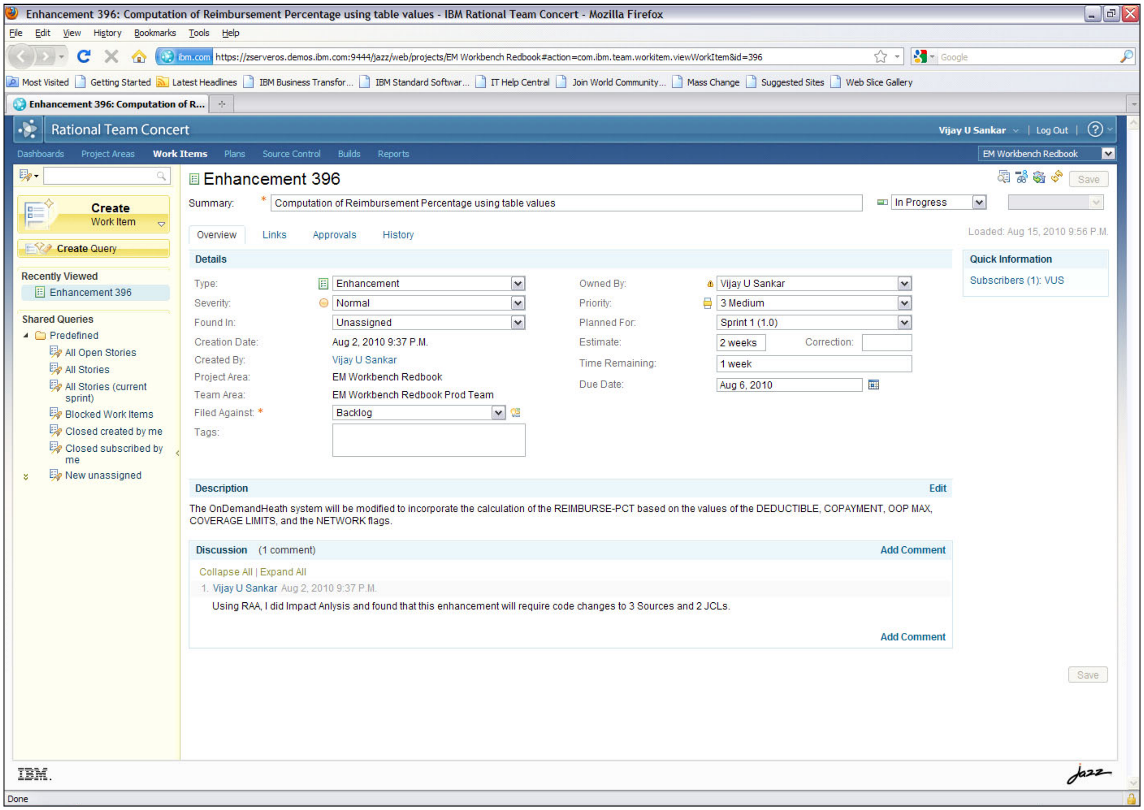


Figure 12-1 RTCz 2.x Work Item

> **RTCz:** All of the RTCz work in Chapter 4 was done using RTCz version 2.x. This was only because the version 3.x release of RTCz (as described in Chapter 3) was not available to this team during development.

IMP:

- The Functional Requirement must be reviewed and signed off by the client.

Traditionally, you bury this statement of modification in program comments. This practice has led to haphazard business rules, as well as task management. Instead, we will add this text into the RTCz repository – to govern this entire process, creating persistent and high-value documentation.

12.4 Impact analysis

Impact analysis is a very important part of a change request, which provides an initial insight to the level of complexity and the amount of effort required to achieve the change. Using RAAi, we perform an impact analysis on the variable REIMBURSE-PCT, because this is the data-element whose computational logic is going to be modified.

Impact analysis results shows that there are three source programs and one JCL member impacted, apart from a number of data-elements, which need to be further analyzed (Figure 12-2 and Figure 12-3).

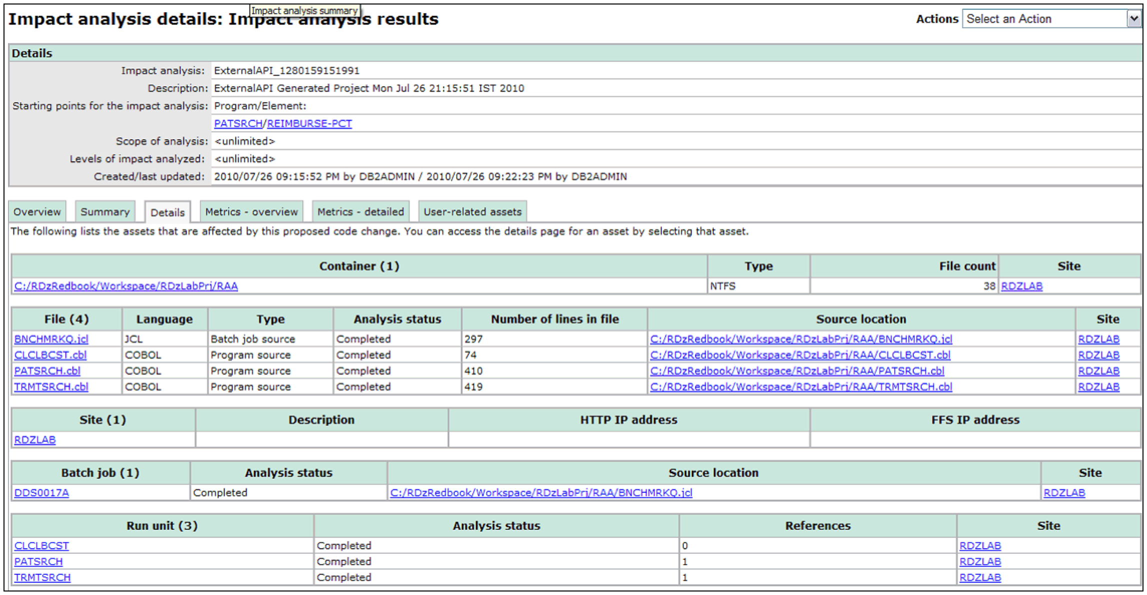


Figure 12-2 Impact Analysis

Entry point (1)	Program	Language	Type	Source location	Site
CLCLBCST	CLCLBCST	COB	primary	C:/RDzRedbook/Workspace/RDzLabPrj/RAA/CLCLBCST.cbl	RDZLAB

Literal (4)	References
1.28	9
2.2	8
70	7
80	25

Program (3)	Language	Analysis status	Number of lines in program	Source location	Site
CLCLBCST	COB	Completed	74	C:/RDzRedbook/Workspace/RDzLabPrj/RAA/CLCLBCST.cbl	RDZLAB
PATSRCH	COB	Completed	521	C:/RDzRedbook/Workspace/RDzLabPrj/RAA/PATSRCH.cbl	RDZLAB
TRMTSRCH	COB	Completed	554	C:/RDzRedbook/Workspace/RDzLabPrj/RAA/TRMTSRCH.cbl	RDZLAB

Data element (13)	Site	Program	Program source location	Declared line	Level	Language	Type	Physical length	Logical length	Scale	Declared in copy?
ANCILLARY-COSTS	RDZLAB	CLCLBCST	C:/RDzRedbook/Workspace/RDzLabPrj/RAA/CLCLBCST.cbl	34	5	COB	NUMB	4	7	2	N
CALC-COSTS-REC	RDZLAB	CLCLBCST	C:/RDzRedbook/Workspace/RDzLabPrj/RAA/CLCLBCST.cbl	24	1	COB	GRP	46			N
CALC-COSTS-REC	RDZLAB	PATSRCH	C:/RDzRedbook/Workspace/RDzLabPrj/RAA/PATSRCH.cbl	164	1	COB	GRP	46			N
EQUIPMENT-COSTS	RDZLAB	CLCLBCST	C:/RDzRedbook/Workspace/RDzLabPrj/RAA/CLCLBCST.cbl	37	5	COB	NUMB	5	9	2	N
MISC-FIELDS	RDZLAB	CLCLBCST	C:/RDzRedbook/Workspace/RDzLabPrj/RAA/CLCLBCST.cbl	20	1	COB	GRP	6			N
NET-PATIENT-COSTS	RDZLAB	CLCLBCST	C:/RDzRedbook/Workspace/RDzLabPrj/RAA/CLCLBCST.cbl	36	5	COB	NUMB	5	9	2	N
PATIENT-COPAY	RDZLAB	CLCLBCST	C:/RDzRedbook/Workspace/RDzLabPrj/RAA/CLCLBCST.cbl	31	5	COB	NUMB	2	3		N
PATIENT-DEDUCTIBLE-REM	RDZLAB	CLCLBCST	C:/RDzRedbook/Workspace/RDzLabPrj/RAA/CLCLBCST.cbl	30	5	COB	NUMB	2	4		N
PROCEDURE-BASE-COST	RDZLAB	CLCLBCST	C:/RDzRedbook/Workspace/RDzLabPrj/RAA/CLCLBCST.cbl	33	5	COB	NUMB	5	9	2	N
REIMBURSE-PCT	RDZLAB	CLCLBCST	C:/RDzRedbook/Workspace/RDzLabPrj/RAA/CLCLBCST.cbl	32	5	COB	NUMB	2	3		N
REIMBURSE-PCT	RDZLAB	PATSRCH	C:/RDzRedbook/Workspace/RDzLabPrj/RAA/PATSRCH.cbl	172	5	COB	NUMB	2	3		N
TEMP-COST	RDZLAB	CLCLBCST	C:/RDzRedbook/Workspace/RDzLabPrj/RAA/CLCLBCST.cbl	21	5	COB	NUMB	6	11	2	N
VENIPUNCTURE-COSTS	RDZLAB	CLCLBCST	C:/RDzRedbook/Workspace/RDzLabPrj/RAA/CLCLBCST.cbl	35	5	COB	NUMB	4	7	2	N

Figure 12-3 Impact Analysis results

From the RAAi Impact Analysis results (Figure 12-3), we get a detailed list of all the impacted components in the application. This helps in determining the scope of the Change Request in terms of schedule and effort.

Results: Clearly, in ISPF it not possible to get this kind of impact analysis results. In ISPF we are manually searching the variable in all the components and deducing the results.

12.5 Change design

The RAA impact analysis results from Figure 12-3 are taken as input and further analysis is performed to prepare a detailed technical design. A few of the critical points of the design for the foregoing change are discussed here. Notice that these next few sections are very detailed and describe procedurally oriented tasks at the software practitioner level.

12.5.1 Stored procedure

The business logic to compute the patient's insurance REIMBURSE-PCT (the percentage of a hospital charge covered by the patient's plan), is the crux for this change. The new logic will be implemented using a DB2 stored procedure because of the following reasons:

- Using RDz's Search option (see Figure 12-4, Figure 12-5, and Figure 12-6), we find that all the variables that determine the value of REIMBURSE-PCT (DEDUCTIBLE, COPAYMENT, OOP MAX, COVERAGE LIMITS, and the NETWORK flag IN-NETWORK-FLAG) are declared in the DCLGEN copybook HLTHPLAN (that is, available in the DB2 table HEALTH_PLAN). (Figure 12-4).

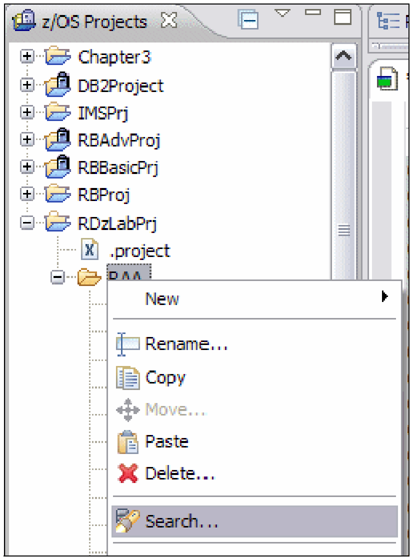


Figure 12-4 RDz Search option

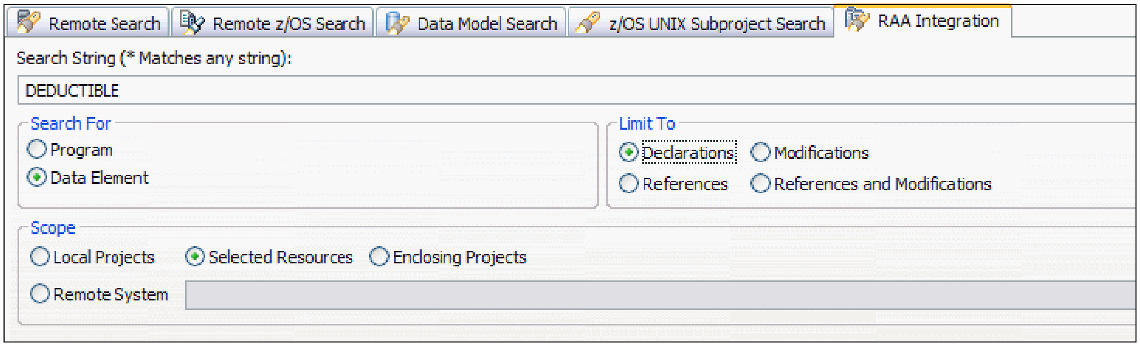


Figure 12-5 Specifying Search attributes for DEDUCTBLE

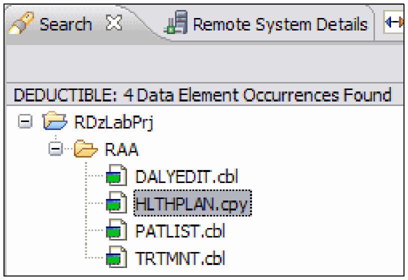


Figure 12-6 Search results for DEDUCTIBLE displayed

- Using a stored procedure provides the following advantages:
  – Reducing the traffic of information across the communication network
  – Splitting the application logic and encouraging an even distribution of the computational workload
  – Providing an easy way to call a remote program

A new stored procedure will be created using the RDz wizard. The stored procedure PCTPROC will have the following logic:

- Input Parameters: PLAN-ID and IN-OUT-NETWORK
- Output Parameters: REIMBURSE-PCT and SQLCODE

The database access component of the new business logic is shown in Example 12-2.

Example 12-2   Business logic

```
EXEC SQL
            SELECT COPAYMENT, COVERAGE_LIMITS, DEDUCTIBLE,
                   IN_NETWORK_REQ, OOP_MAX
            INTO
             :COPAYMENT              ,
             :COVERAGE-LIMITS        ,
             :DEDUCTIBLE             ,
             :IN-NETWORK-REQ         ,
             :OOP-MAX
            FROM DDS0001.HEALTH_PLAN
            WHERE PLAN_ID = :PLANID
           END-EXEC.
           IF SQLCODE = +100
             MOVE 10 TO REIMBURSEPCT.
IF SQLCODE = +0
             IF DEDUCTIBLE > 0 AND
                     COPAYMENT > 0  AND
                      OOP-MAX < 1000  AND
                      COVERAGE-LIMITS > 10000 AND
                      IN-NETWORK-REQ = 'Y' AND INOUTNETWORK = 'Y'
                    MOVE 80 TO REIMBURSEPCT
                 ELSE
                  IF (DEDUCTIBLE > 0 AND COPAYMENT = 0)  OR
                   (DEDUCTIBLE = 0 AND COPAYMENT > 0) AND
                          OOP-MAX < 1000  AND
                          COVERAGE-LIMITS > 10000 AND
                       IN-NETWORK-REQ = 'Y' AND INOUTNETWORK = 'Y'
                     MOVE 60 TO REIMBURSEPCT
                 ELSE
                    IF IN-NETWORK-REQ = 'Y' AND INOUTNETWORK = 'N'
                         MOVE 40 TO REIMBURSEPCT
                    ELSE
                         MOVE 30 TO REIMBURSEPCT.
           MOVE SQLCODE TO W-SQLCODE.
           MOVE W-SQLCODE TO SQLCODEOUT.
```

12.5.2 Modification to the source modules

Using Rational Asset Analyzer Integration (RAAi) Impact Analysis, we have identified which source modules will need modification. In the analysis, we ensure that the input parameters to the stored procedure are available in the program. See Chapter 5, "Rational Asset Analyzer" on page 199 for details.

Using the RAAi features, Program Tree, Control Flow, Data Flow, and RAAi Search, we analyze the source to trace the flow of several values necessary to come up with the following changes to the source modules.

1. **Changes to the COBOL program: MSTRUPDT**
   a. Comment out the existing logic which hard codes the value of REIMBURSE-PCT.
   b. Code new logic to populate the input parameters to the stored procedure is determined as follows:
      - **PLAN-ID** → Using the RAAi Integration Search feature looking for modification to PLAN-ID and finding that the PLAN-ID gets its value from the variable INS-IDENT-NBR in the input data set PATINS (Patient Insurance). On double-clicking the search result, opens the statement in MSTRUPDT as shown in Figure 12-7, Figure 12-8, and Figure 12-9.

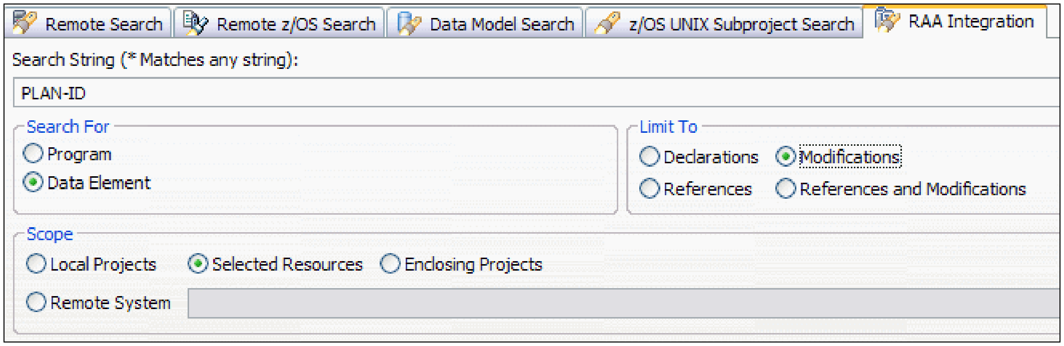


Figure 12-7 Specifying Search attributes for PLAN-ID

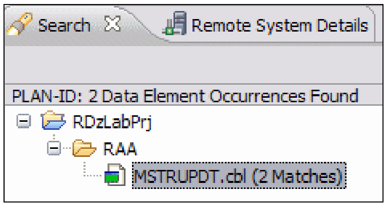


Figure 12-8 Search results for PLAN-ID displayed

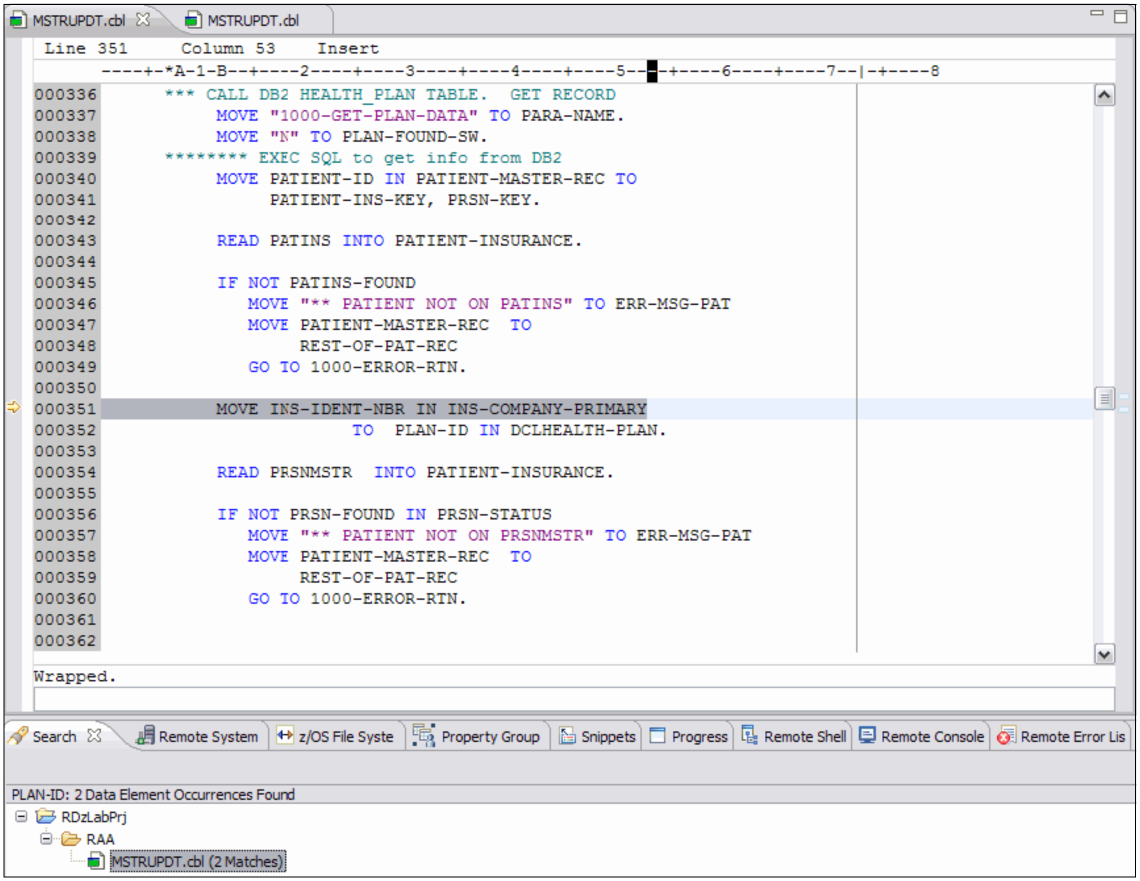


Figure 12-9 Double-clicking the search result opens the component

- **IN-OUT-NETWORK** → Using the RAAi Search option, we find that IN-OUT-NETWORK is one of the values in the input data set PATMSTR (Patient Master).

c. Logic to call to stored procedure PCTPROC to get the value of REIMBURSE-PCT

d. Logic to call to DB2 error routine in case of a DB2 error.

2. PATSRCH and TRMTSRCH

Using RAAi, we found that the input parameter PLAN-ID is not available in the programs PATSRCH and TRMTSRCH. RAA also showed that PLAN-ID is the primary key of the VSAM file and declared in the copybook PATINS – which is used in several other programs.

So, apart from the four changes mentioned for MSTRUPDT, the additional changes to these programs are as follows:

1. Include the copybook SQLCA, because both the programs are non-DB2 and now we are changing it into a DB2 COBOL program.
2. Include the DCLGEN for the DB2 table HEALTH_PLAN, HLTHPLAN.
3. Logic to handle the file PATINS as input to the programs.
4. Logic to handle the error in case the PLAN-ID is not found in the input.

12.5.3 Modifications to JCL

RAA also documents which JCL will be affected by the change:

1. REBUILDQ:

   The compile and Link Edit steps for PATSRCH and TRMTSRH will be modified to include the DB2 parameters.

2. BNCHMRKQ2:

   The PATSRCH and TRMTSRCH steps will be modified to incorporate the PATINS file as input. Also the step will now include a control card or SYSIN card for DB2 parameters.

We will be creating the stored procedure using the Data Perspective in RDz. This requires a set of authorizations that will be provided by the DBAs. To provide DB2 development features, the workbench accesses DB2 system catalog tables. The user ID that the workbench uses must have the following privileges:

- CONNECT: To connect to the Database system, in this case it is DB1S.
- CREATE PROCEDURE: To create the stored procedure.
- BINDADD: To create the package
- CREATEIN privilege on the desired collection ID: To build and install the stored procedure in the host system. Here the Collection ID is DSS0001C.

12.5.4 Test plan

In the Design phase itself, we prepare the Test Plan (for any change, including trivial changes). in parallel, based on the functional requirement and the design. The test plan ideally contains the entry and exit criteria for the change being implemented. RAAi can also help with the test plan – as it surfaces file and database dependencies for a program.

A sample test case is shown in Table 12-1.

Table 12-1 Sample Test Case for stored procedure PCTPROC

Sl No	Action involved	File/database table	Expected result	Actual result	Pass/fail
	Run the stored procedure PCTPROC with the input as 'GBINS-2FD-T00IX 8I-00' for PLAN-ID and ‘Y’ for IN-OUT-NETWORK	HEALTH_PLAN table	The stored procedure must return ‘30’ for REIMBURSE-PCT		

IMP:

- The Design and Test Plan must undergo the review according to the Quality process.
- The Detailed Technical Design Document must be signed-off by the client.
- The Test Case document must be signed-off by the client.

12.6 Development

After all the necessary signoffs are received and the DBA request for stored procedure authorization has been completed for the development environment, the development of the change is started. The development phase involves coding the changes and then testing them until the exit criteria are satisfied.

12.6.1 Coding

In this section we discuss various considerations regarding the coding.

Connections required to z/OS when creating stored procedures

Creating COBOL and PL/I OS/390 stored procedures using RDz requires two connections to z/OS:

- One connection establishes a link between RDz and the DB2 Universal Database server on z/OS so that DB2 catalog information about tables, columns, stored procedures, and so on can be displayed and operations (drop, import, sample contents, and so on) on these DB2 assets can be performed. This connection is created using the RDz Data perspective and Data Source Explorer.

- The other connection establishes a link between RDz and z/OS so that the generated stored procedure can be stored on and read from z/OS. This connection can be created using the RDz z/OS Systems perspective (Figure 12-10).

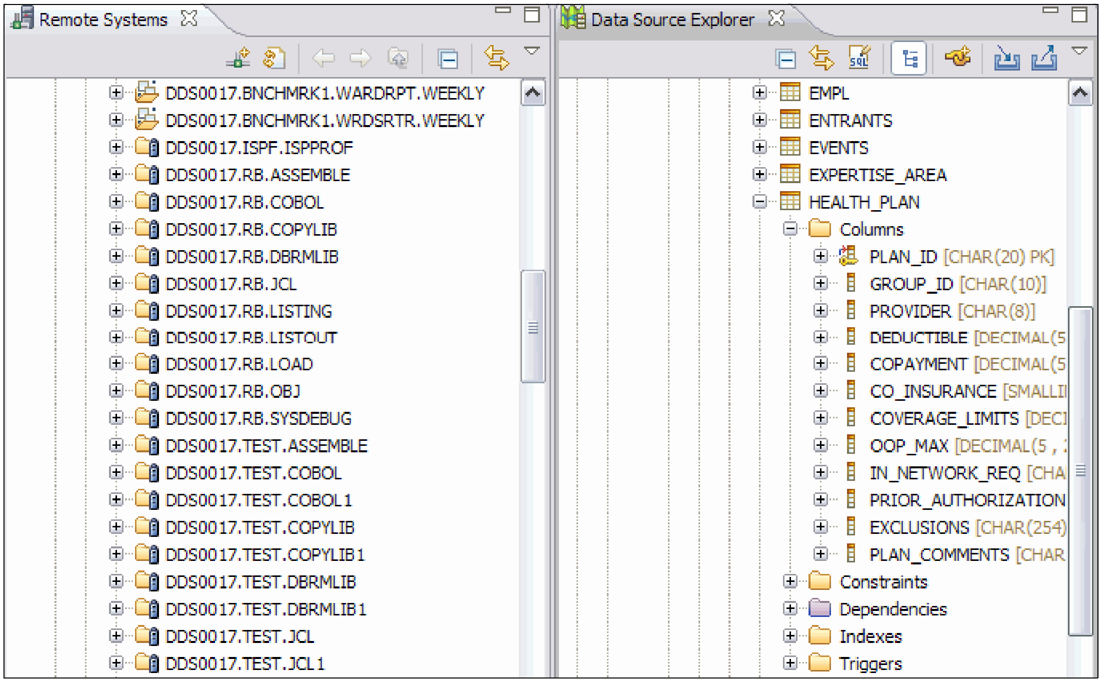


Figure 12-10 Data perspective with the z/OS Systems view

Tips for creating stored procedures

The following techniques will help you to develop more efficient and useful stored procedures:

- Pass all the input data and parameters from the client application to the stored procedure at invocation time.
- Return all the result data and output parameters to the client application only when the stored procedure is complete.
- Define the input and output parameters in both the client application and the stored procedure. The stored procedure must expect the passed parameters

Creating a COBOL stored procedure

To create a COBOL stored procedure, use the Data Project Explorer view of the Data Perspective after creating a new Project (Figure 12-11).

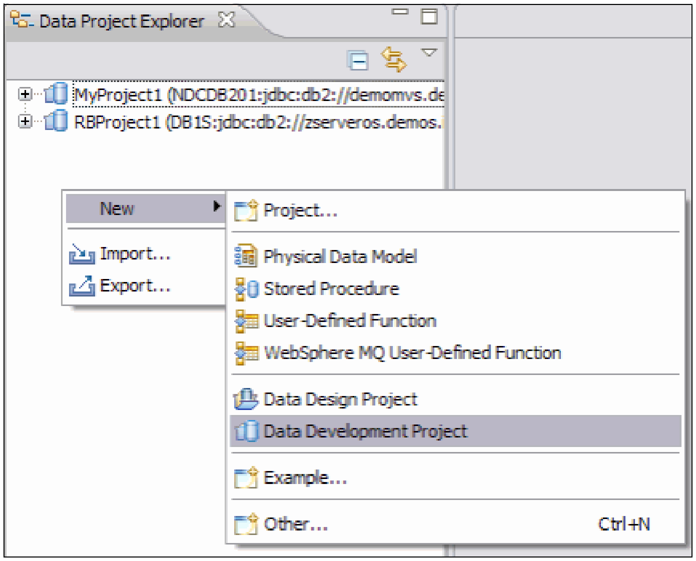


Figure 12-11 Creating a new Data Development Project

Then follow these steps:

1. Expand the project and create a new stored procedure as shown in Figure 12-12.

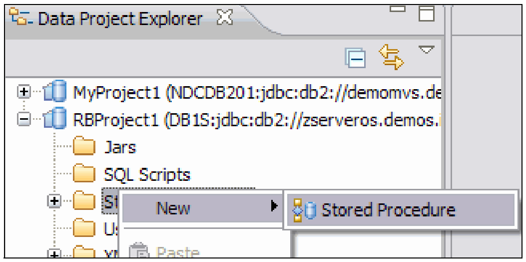


Figure 12-12 Creating a new COBOL stored procedure in the Data Project

2. Specify the initial information such as project name, language and stored procedure name. In the Source Location dialog, specify the field values as shown in Figure 12-13.

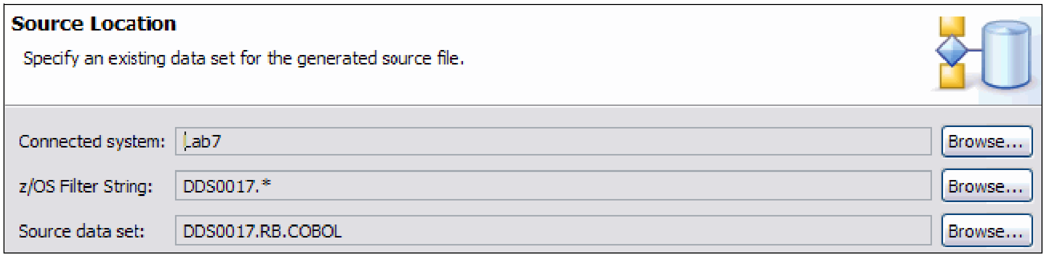


Figure 12-13 Figure 13-12 Source Location details for the stored procedure

3. Specify the SQL statement that will be used in the stored procedure (Figure 12-14).

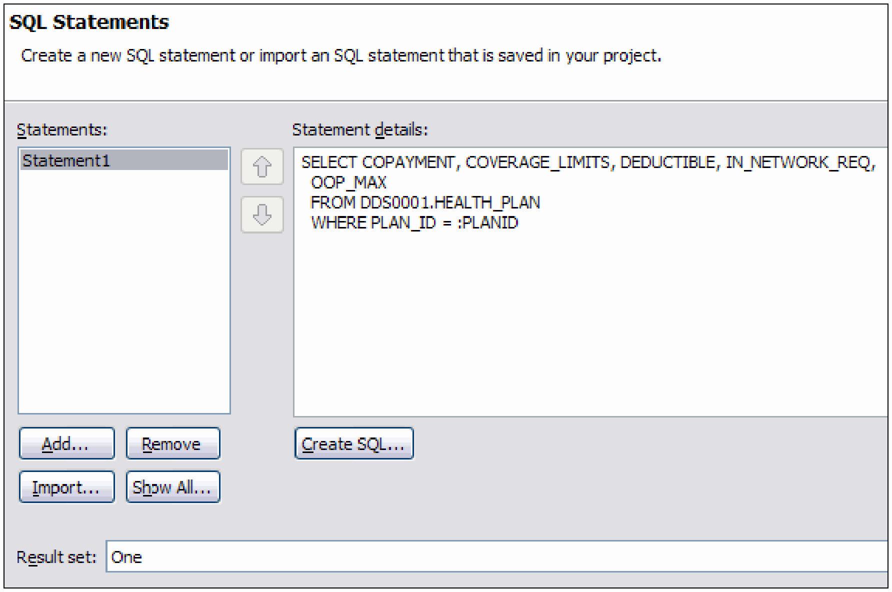


Figure 12-14 SQL statement to be used in stored procedure

4. Specify the characteristics of the parameter and click **Apply** for each parameter. When you are finished, click **OK**. Define the following parameters for the stored procedures to implement our example, as shown in Table 12-2.

Table 12-2 Input and Output parameters for the stored procedure

Parameter Mode	Name	SQL Type	Length
In	PLANID	CHAR	20
In	INOUTNETWORK	CHAR	1
Out	REIMBURSEPCT	SMALLINT	
Out	SQLCODEOUT	INTEGER	

> **Tip:** Do not enter invalid COBOL names for the parameters. For example, CUST_ID will be accepted when creating the stored procedures parameters, but when the COBOL for a legitimate COBOL variable named: CUST-ID is generated, the generated data names will be invalid, because CUST-ID will get transformed to: CUST_ID – and CUST_ID (with an underscore) is an invalid COBOL name. Also do not use invalid DB2 names like CUST-ID. Our suggestion is not to use any special characters as parameter names

5. When the New COBOL stored procedure dialog box opens, click **Next**. Enter the Collection ID into the entry field. The collection ID is used for binding the package of the stored procedure.
6. Click **Advanced**. In the z/OS Options window:
   – Enter your WLM environment, for example, DSNAWLM. This is where the stored procedure will run. The stored procedure will run in the DB2 established stored procedure address space if you do not enter a WLM environment name.

– Select **DB2 for External security.** This option specifies how the stored procedure interacts with an external security product, such as RACF, to control access to non-SQL resources (Figure 12-15).

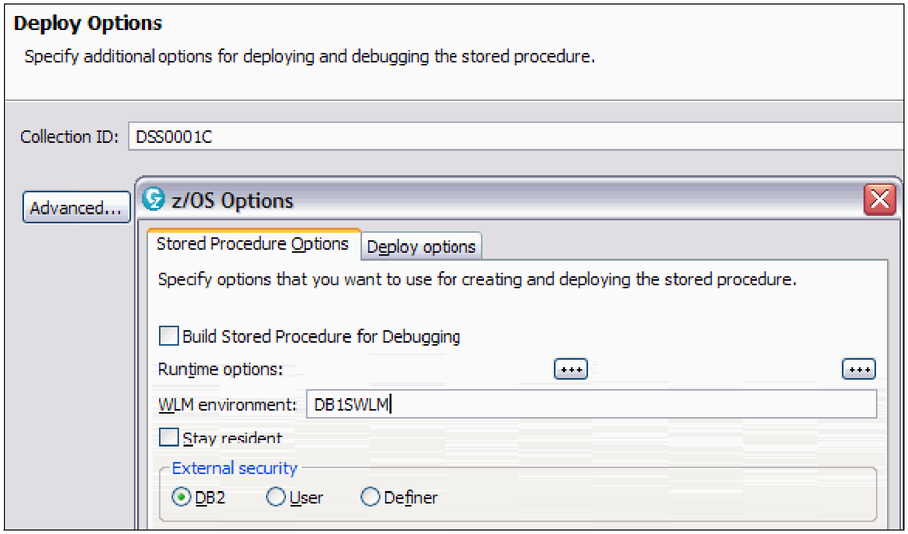


Figure 12-15 Specifying WLM environment

7. On the Summary page of the “New COBOL Stored Procedure” window, click **Finish**. The code is generated. The generated stored procedure source code appears in the Editor view, as shown in Figure 12-16. You must be connected to z/OS, because the code will be generated at the z/OS database.

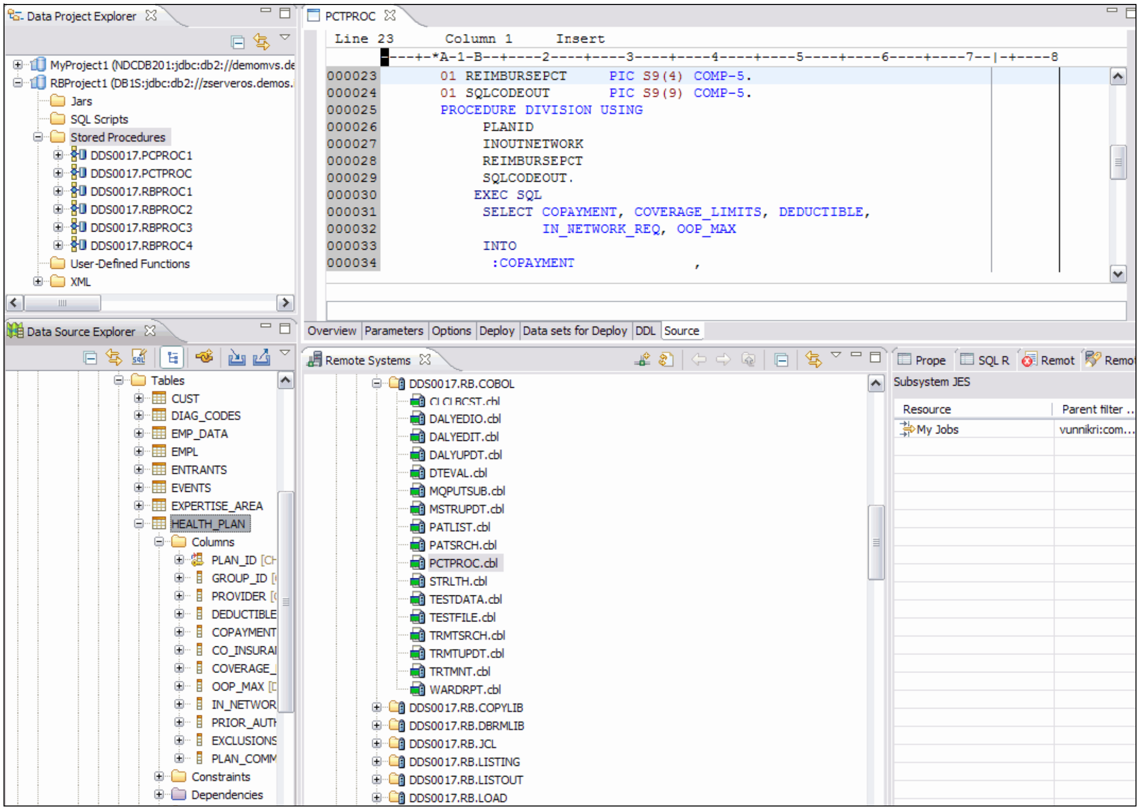


Figure 12-16 COBOL stored procedure generated

8. In the Editor View, select the various tabs, **Overview**, **Parameters**, **Options**, **Build**, **Data sets for Build**, and **DDL**, to ensure that all is set for your environment, as shown in Figure 12-17.

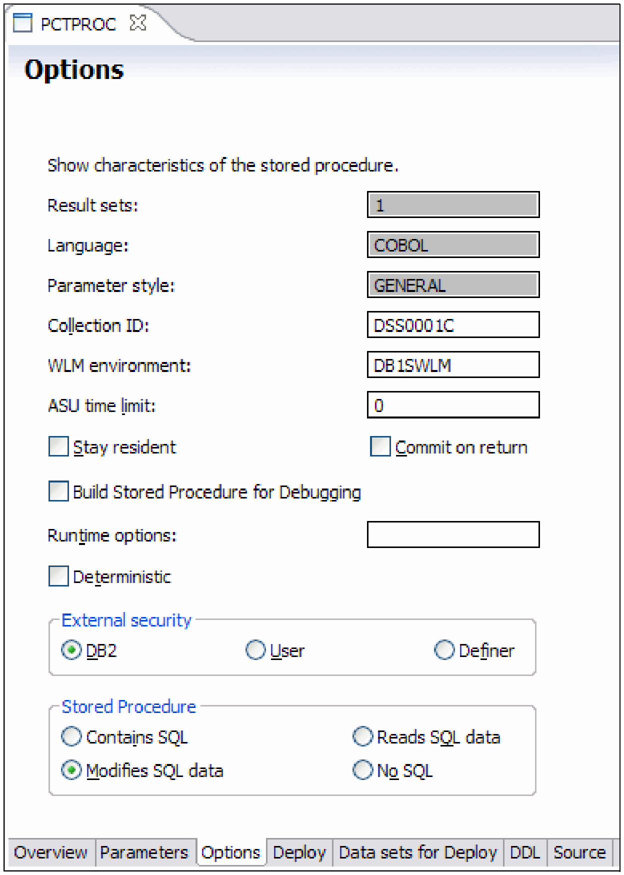


Figure 12-17 Stored procedure Options

Options that you can change here include these:

- **ASU time limit**: Specifies the total amount of processor time, in CPU service units, that a single invocation of a stored procedure can run.
- **Stay resident**: Specifies whether the stored procedure load module is to remain resident in memory when the stored procedure ends.
- **Commit on return**: Indicates whether DB2 commits the transaction immediately on return from the stored procedure.
- **Build Stored Procedure for Debugging**: Select these options to generate the runtime options for debugging.
- **Runtime options**: IBM Language Environment® runtime options to control certain aspects of the program processing.
- **Deterministic**: Specifies whether the stored procedure returns the same result from successive calls with identical input arguments.
- **External security**: Specifies how the stored procedure interacts with an external security product, such as RACF, to control access to non-SQL resources.
- **Stored procedure**: Indicates whether the stored procedure can execute any SQL statements and, if so, what type. Use the NO SQL option if your stored procedure does not contain SQL statements. The build utility will invoke the SQL statement coprocessor and try to bind a package unless the NO SQL option is set.

9. At this point, we have created COBOL skeleton code in the z/OS system, and now, we modify it according to our design. Note that the COBOL skeleton generated has all the parameters, the linkage section, and the procedure division created according to our input in the pervious windows.

Building the COBOL stored procedure

Before a stored procedure can be invoked from the DB Servers view or any other calling program, it must be built on the remote system. The process for building the DB2 stored procedure is as follows:

1. Click the **Data sets for Build** tab in the Editor view (Figure 12-18). Check if the values are correct. Some of the options might already have default values, which are obtained from the z/OS Properties. You can change these default property values if needed. Changing the values in these fields does not affect the z/OS build properties. Changing these values affects only the stored procedure build properties.

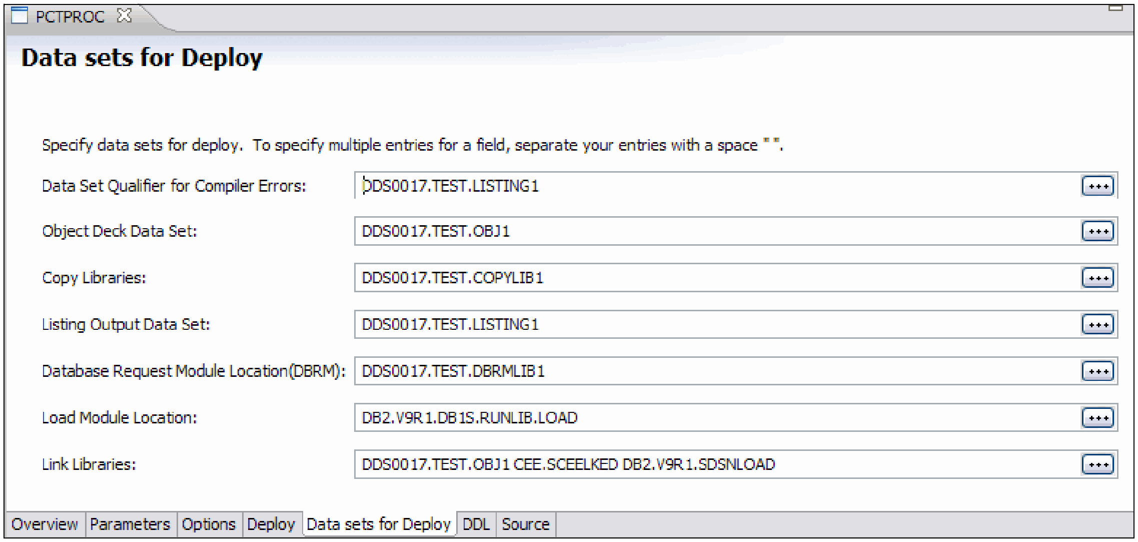


Figure 12-18 Data sets used for stored procedure build

2. Optionally, you can do a Local Syntax Check which might avoid unnecessary compilations at the z/OS database.
3. To build the stored procedure, right-click the stored procedure in the Data Project Explorer view and select **Deploy** as shown in Figure 12-19.

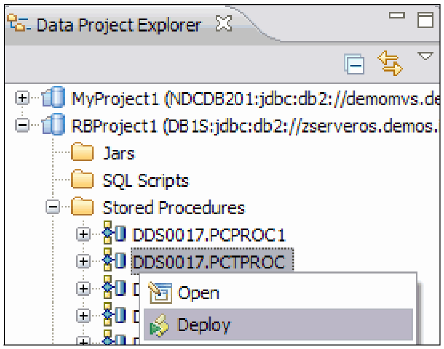


Figure 12-19 Deploying the stored procedure

The build utility is now invoked on the remote system and builds the executable form of the stored procedure. This process includes the compilation of the source code, the linkage of the load module, the binding of the DB2 package, and the registration of the stored procedure in the DB2 catalog.

The build process is displayed in the SQL Result view with the status “In progress” until it is completed. The status in the SQL Result view changes to “Success” or “Failure” when the build process is completed.

When execution is successful, a short summary of the build process is displayed. If the build process was not successful, information about the failure will be displayed in the SQL Result view. Figure 12-20 shows the successful build of our example.

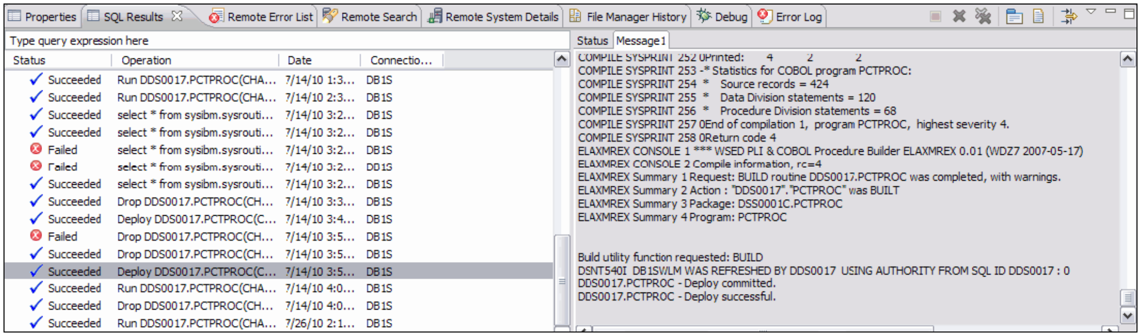


Figure 12-20 Shows the Deploy was successful

> **RDz:** In ISPF, it is a very tedious and complex process to build and deploy a stored procedure and it requires a dedicated DBA involvement to achieve it. Whereas, using RDz, we find that the tool makes the process very easy and time-saving, provided that all the authorizations are in place.

Code modification to the existing modules

RTCz is used here to do the source control management. After connecting to the Repository and creating a Repository Workspace, we use the following steps to retrieve the components from Production to do the code modifications:

1. Create an MVS Subproject in RDz, where the checked-out components will be retrieved to (Figure 12-21).

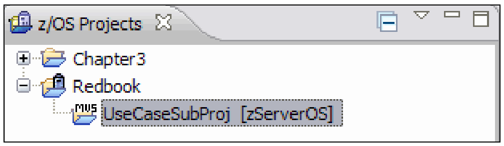


Figure 12-21 MVS Subproject created

2. Switch to the Work Items Perspective and check-out the components to be modified using the ‘Loading zFiles to zOS…’ feature of RTCz (Figure 12-22).

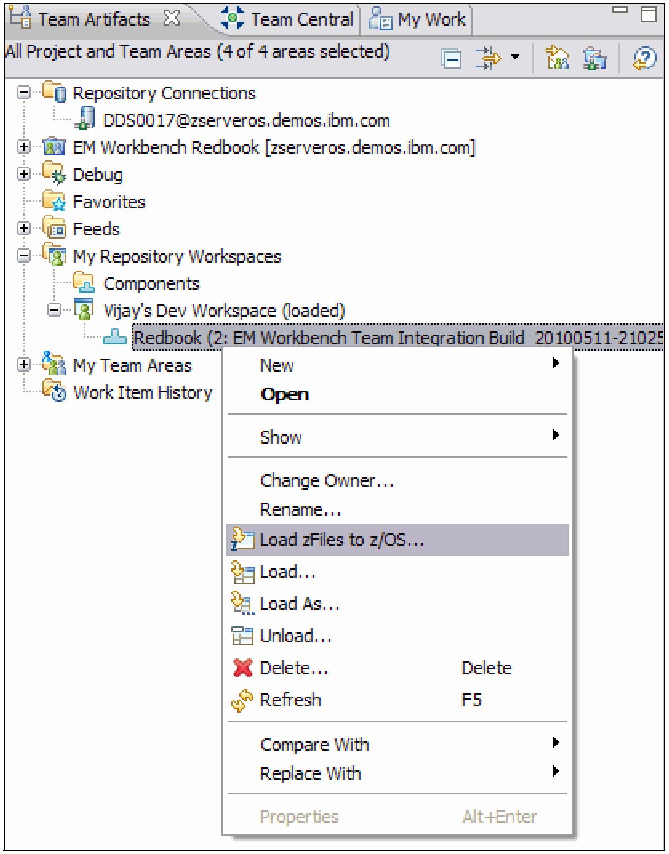


Figure 12-22 Option to Check-Out the components in RTCz

3. The components MSTRUPDT.cbl, PATSRCH.cbl, TRMTSRCH.cbl, REBUILD.jcl, and BNCHMRK.jcl are selected (Figure 12-23).

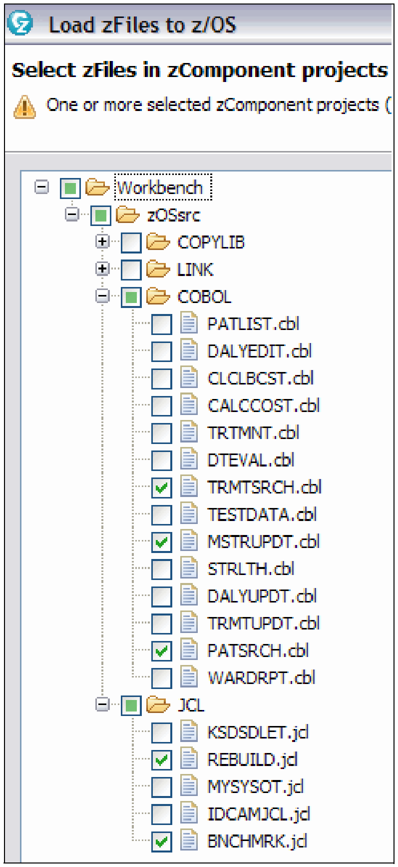


Figure 12-23 Selecting components to be checked out

4. Specify the parameters for Build Definition like the dataset prefixes to be created in the MVS Subproject (Figure 12-24).

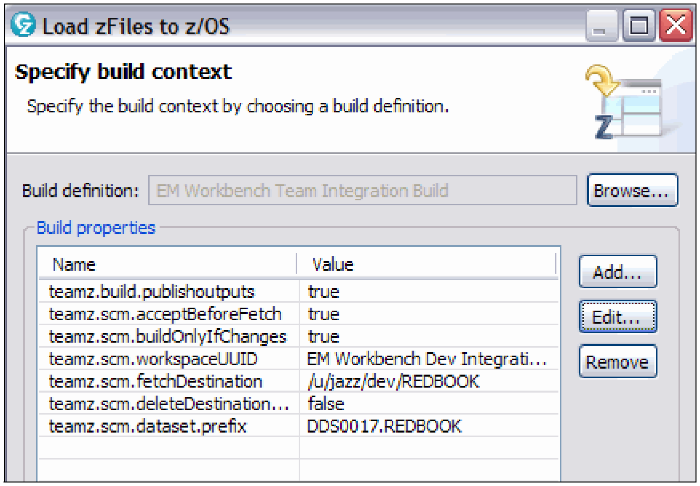


Figure 12-24 Specifying Build Context

5. After the steps in the wizard are completed, the components are checked out to the MVS Subproject specified (Figure 12-25).

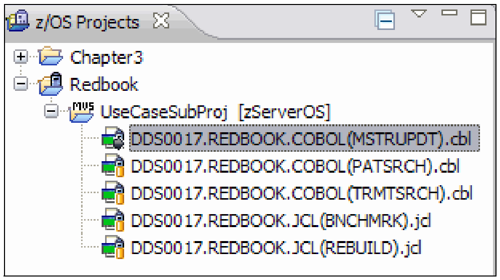


Figure 12-25 Components checked out into the MVS Subproject

MSTRUPDT COBOL program

Follow these steps for the coding:

1. Declare the new working storage variables to be used.
2. Using the Comment feature of RDz context menu in LPEX Editor, we comment out the code logic that will be replaced with the new logic, in the paragraphs 3000-CALCULATE-TREATMENT-COSTS, 4000-CALCULATE-EQUIPMENT-COSTS, and 1000-GET-PLAN-DATA (Figure 12-26).

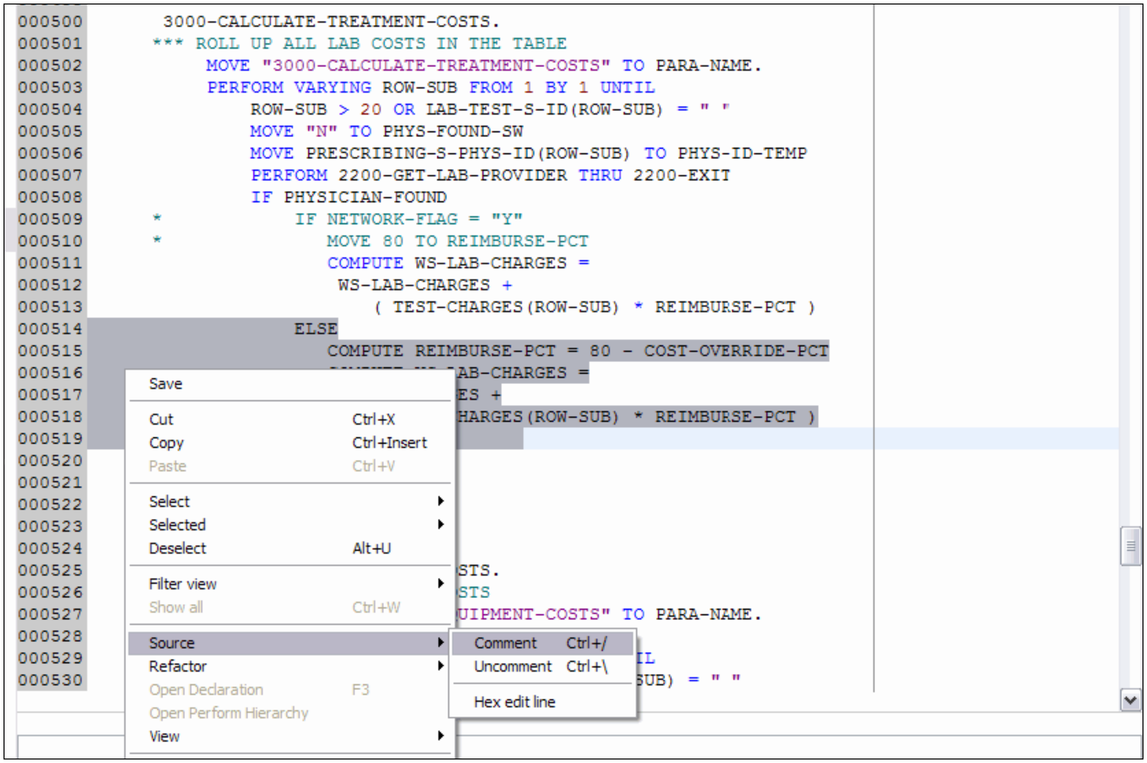


Figure 12-26 Option to comment out a number of lines together

3. Code the logic to populate the input parameters and the call to stored procedure in the paragraph with the help of various features available in LPEX Editor such as content-assist, real-time error messages, and so on, shown in Figure 12-27.

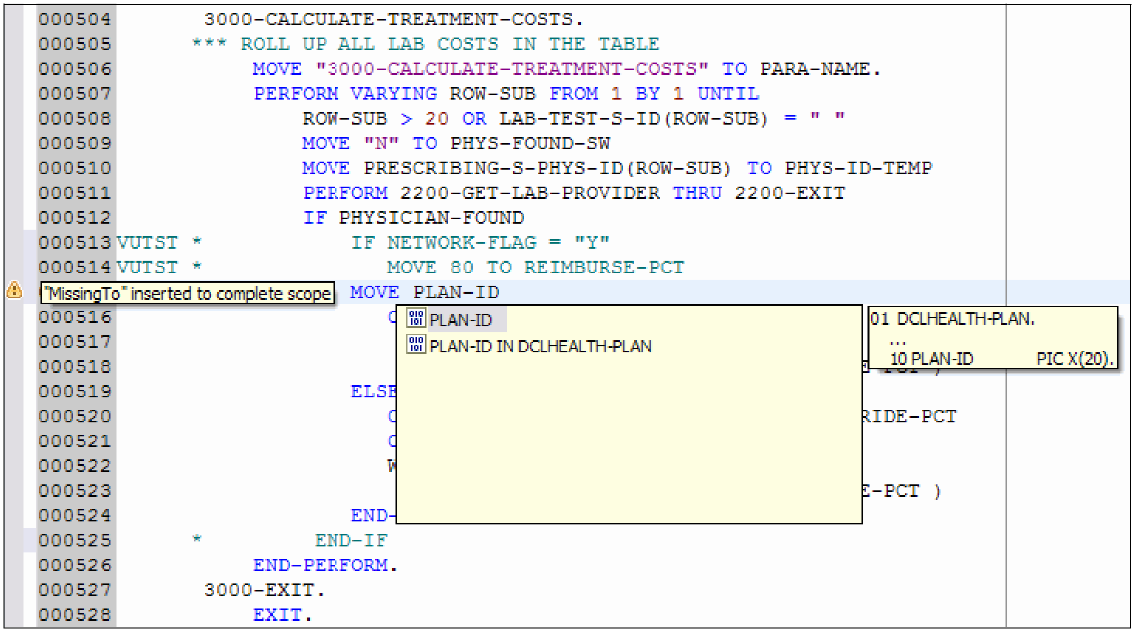


Figure 12-27 Shows different real-time features in RDz

Here is the sample final code change to the paragraph 3000-CALCULATE-TREATMENT-COSTS (Figure 12-28).

```
*MSTRUPDT.cbl
 Line 532      Column 54      Insert     41 changes
       ----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+-
       3000-CALCULATE-TREATMENT-COSTS.
      *** ROLL UP ALL LAB COSTS IN THE TABLE
           MOVE "3000-CALCULATE-TREATMENT-COSTS" TO PARA-NAME.
           PERFORM VARYING ROW-SUB FROM 1 BY 1 UNTIL
               ROW-SUB > 20 OR LAB-TEST-S-ID(ROW-SUB) = " "
               MOVE "N" TO PHYS-FOUND-SW
               MOVE PRESCRIBING-S-PHYS-ID(ROW-SUB) TO PHYS-ID-TEMP
               PERFORM 2200-GET-LAB-PROVIDER THRU 2200-EXIT
               IF PHYSICIAN-FOUND
VUTST                 MOVE PLAN-ID IN DCLHEALTH-PLAN TO PLANID
VUTST                 MOVE IN-OUT-NETWORK OF PATIENT-MASTER-REC TO
VUTST                 INOUTNETWORK
VUTST                 EXEC SQL
VUTST                 CALL PCTPROC (:PLANID,
VUTST                               :INOUTNETWORK,:REIMBURSEPCT,
VUTST                               :SQLCODEOUT)
VUTST                 END-EXEC
VUTST *                IF NETWORK-FLAG = "Y"
VUTST *                   MOVE 80 TO REIMBURSE-PCT
VUTST                 MOVE SQLCODEOUT TO SQLCODE
VUTST                 IF SQLCODE = -811 OR 0
VUTST                    MOVE REIMBURSEPCT TO REIMBURSE-PCT
                         COMPUTE WS-LAB-CHARGES =
                           WS-LAB-CHARGES +
                              ( TEST-CHARGES(ROW-SUB) * REIMBURSE-PCT )
VUTST *                ELSE
VUTST *                   COMPUTE REIMBURSE-PCT = 80 - COST-OVERRIDE-PCT
VUTST *                   COMPUTE WS-LAB-CHARGES =
VUTST *                   WS-LAB-CHARGES +
VUTST *                       ( TEST-CHARGES(ROW-SUB) * REIMBURSE-PCT )
VUTST                 ELSE
VUTST                    IF SQLCODE < 0
VUTST                       GO TO 1000-DB2-ERROR-RTN
VUTST                    END-IF
                      END-IF
                   END-IF
           END-PERFORM.
       3000-EXIT.
           EXIT.
```

Figure 12-28 Snapshots of the code changes to MSTRUPDT program

Similarly, we do the code changes for the COBOL programs PATSRCH, TRMTSRCH, and the JCL, REBUILDQ, BNCHMRKQ according to the Design document prepared earlier in the section.

4. On completing the coding and the required reviews, the modified components are checked in to the repository workspace in RTCz using the ‘Pending Changes’ view (Figure 12-29).

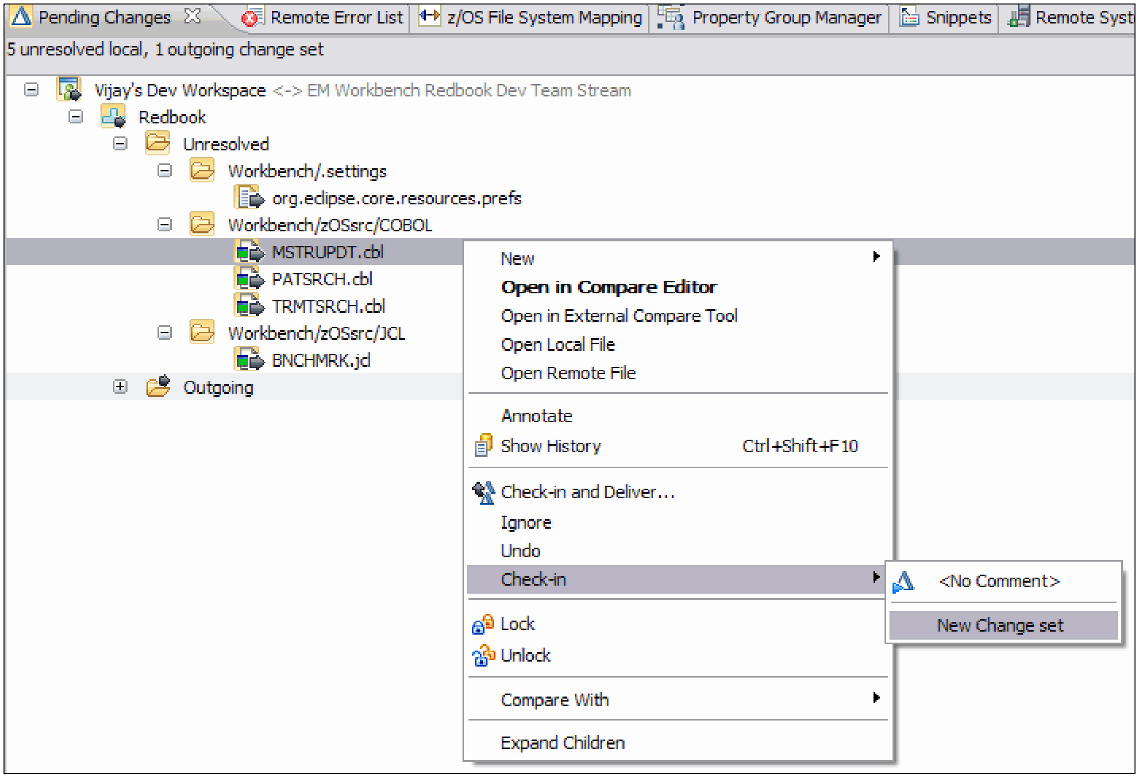


Figure 12-29 Code changes in MSTRUPDT being checked in

> **RDz:** It is very evident here that the numerous features available in RDz make coding a lot more easier and convenient task compared to ISPF. In ISPF, we do not have the options like real-time syntax checks, Content Assist, commenting a bulk of statements, and so on.

IMP:

- Update the revision history of the changed module and tag the code changes according to coding standards specified by the client.
- All the code changes must be properly reviewed according to the Quality process.

12.6.2 Testing

Testing is done with the help of the PD Tools integrated with RDz and Data Perspective, according to the Test Plan prepared during Design phase.

Unit testing

This section describes the testing method:

1. **Testing the stored procedure:**

   a. Using Data Source Explorer view in RDz Data Perspective, we will do a test run of the stored procedure PCTPROC, which was built and deployed in the coding phase (Figure 12-30).

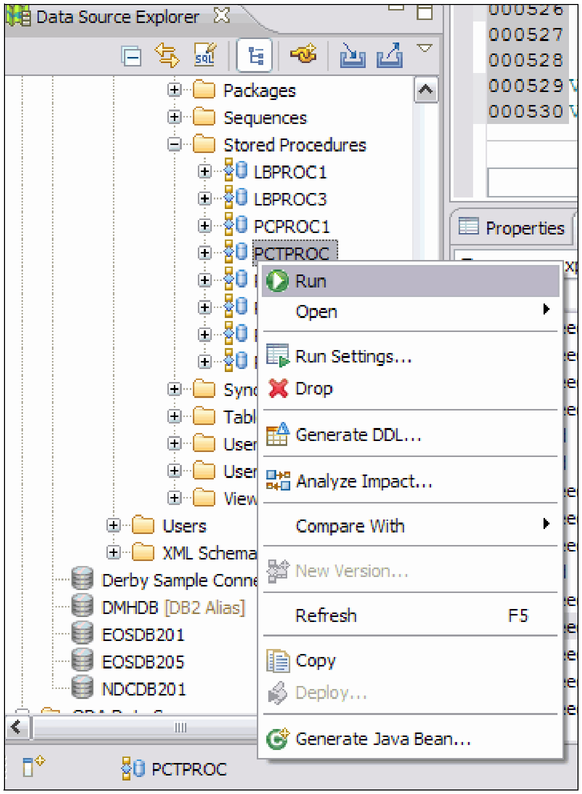


Figure 12-30 Running the stored procedure

A parameter input panel opens if the stored procedure uses input parameters.

b. Specify the parameter values and click **OK** (Figure 12-31).

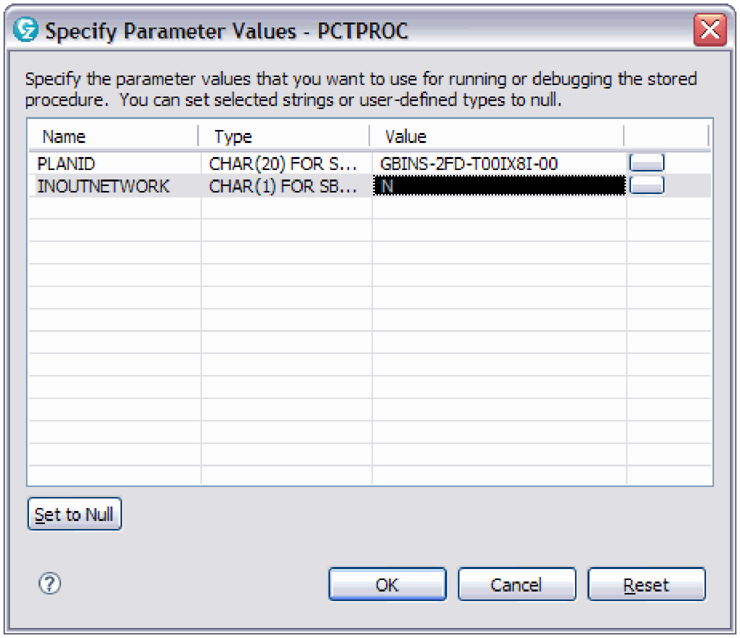


Figure 12-31 Input parameters when running the generated stored procedure

When the run completes, information about the run is displayed in the SQL Result view (Figure 12-32).

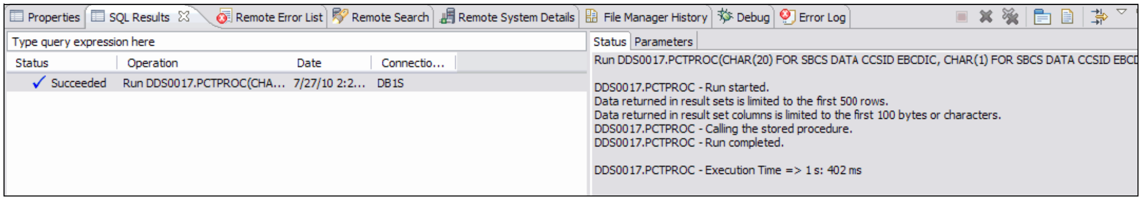


Figure 12-32 Messages when running the stored procedure

The results can be seen in the Parameter tab and the stored procedure PCTPROC successfully returns the value for REIMBURSE-PCT (Figure 12-33).

Status | Parameters

Name	Type	Data type	Value	Value (OUT)
PLANID	INPUT	CHAR	GBINS-2FD-T0...	
INOUTNETWORK	INPUT	CHAR	N	
REIMBURSEPCT	OUTPUT	SMALLINT		40
SQLCODEOUT	OUTPUT	INTEGER		0

Figure 12-33 Results when running the stored procedure

ISPF: In ISPF, in order to run a stored procedure, this requires preparation of JCL with input parameters, which will all be a static process.

2. Preparation of test data:

Using the RAAi Program Diagram, we find out what data is input to the program and what is output from the program for testing (Figure 12-34).

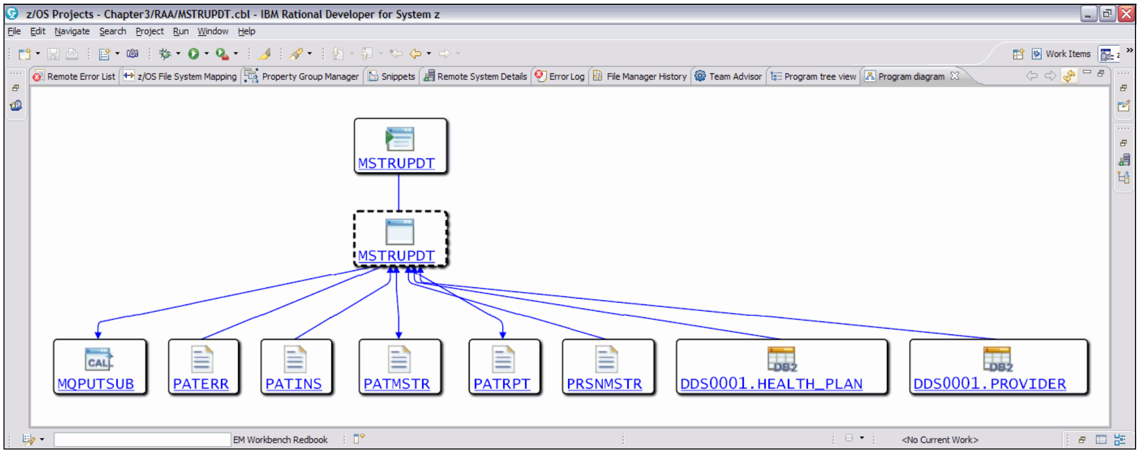


Figure 12-34 RAAi Program Diagram view

Using the File Manager features integrated with RDz, the input data is prepared by allocating the VSAM files and copying test data from the production files:

a. Allocate a new VSAM file TEST.PATMASTR with the parameters like the existing PATMASTR production VSAM file (Figure 12-35).

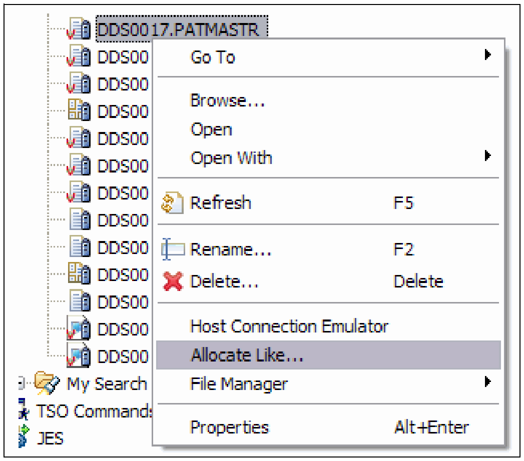


Figure 12-35 Using Allocate Like... option to allocate VSAM dataset

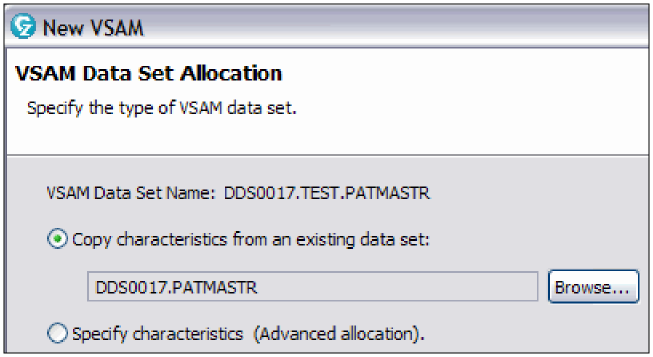


Figure 12-36 Using the parameters of an existing VSAM dataset

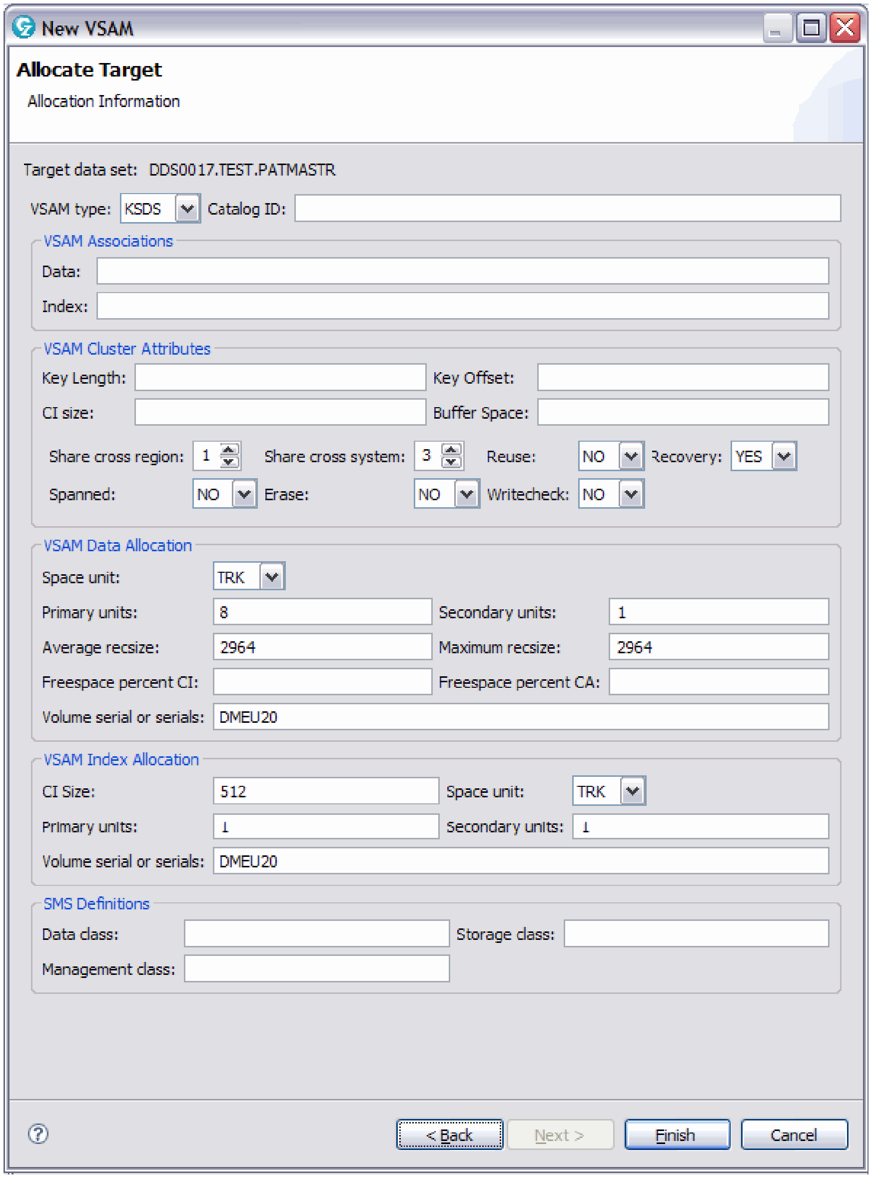


Figure 12-37 Modify any parameter for the target dataset, if required

b. After the VSAM file is created, copy the sample data from the production file (Figure 12-38, Figure 12-39, and Figure 12-40).

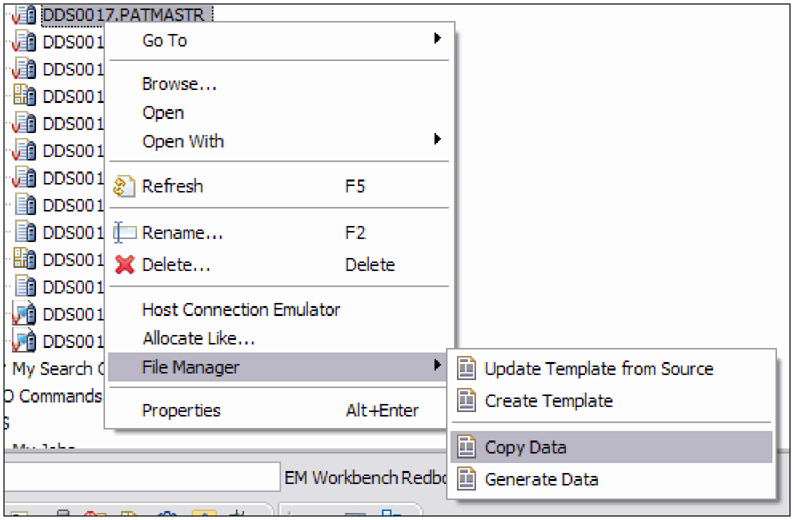


Figure 12-38 Copying data to the new VSAM dataset from an existing one

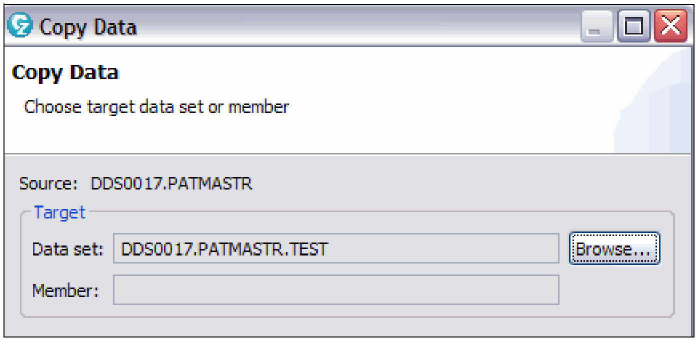


Figure 12-39 Selecting the existing dataset to be copied

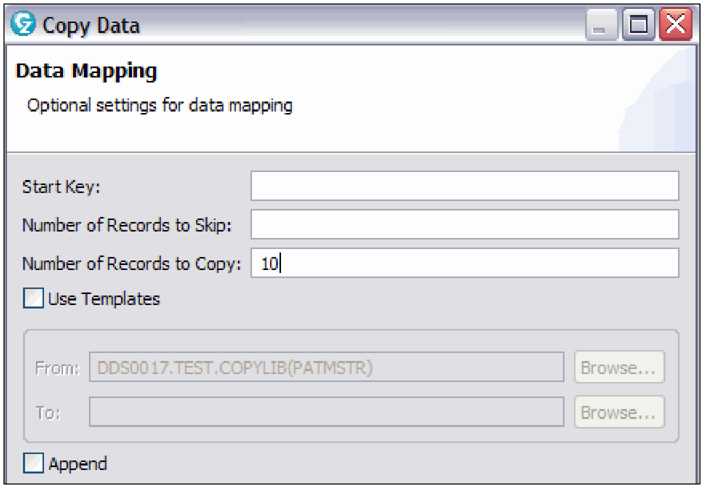


Figure 12-40 Specifying numbers of records to be copied

– Associate the template with the file created and modify the data, if required, according to the test cases, and save it (Figure 12-41 and Figure 12-42).

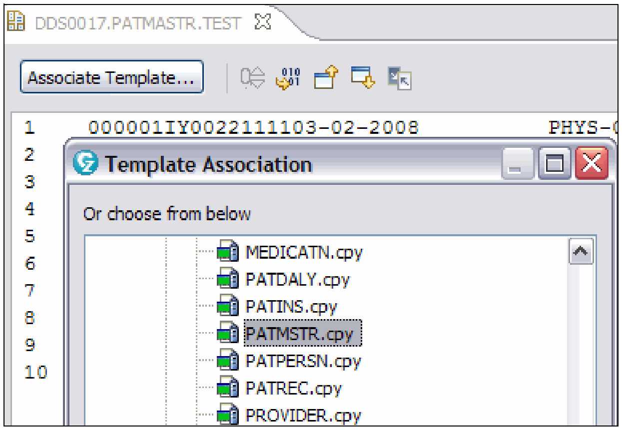


Figure 12-41 Selecting the Template to be used

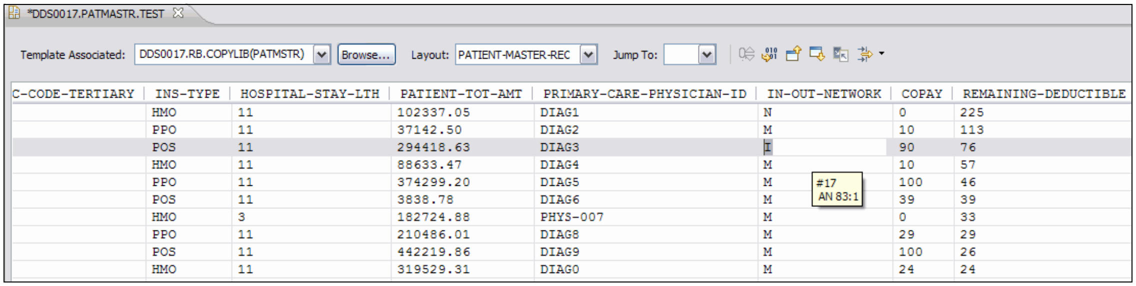


Figure 12-42 New VSAM dataset created

c. Similarly, we allocate and prepare data for other input files: PATINS and PAPERSN VSAM files.

d. Then the output files required are created (it is always a good idea to use GDGs for output files, as we mostly will need to test our job multiple times). There are two GDG output files required by the program: Patient Report (PATRPT) and Patient Error (PATERR) files (Figure 12-43 and Figure 12-44).

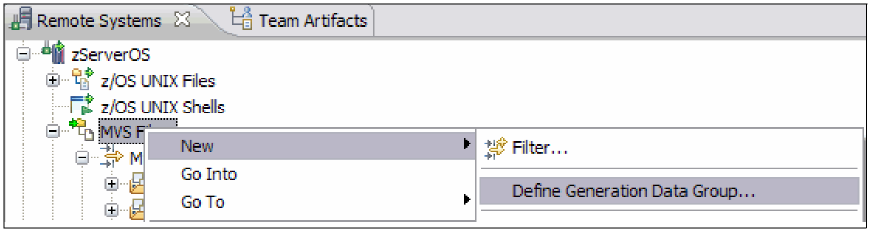


Figure 12-43 Option to create new GDG base in RDz

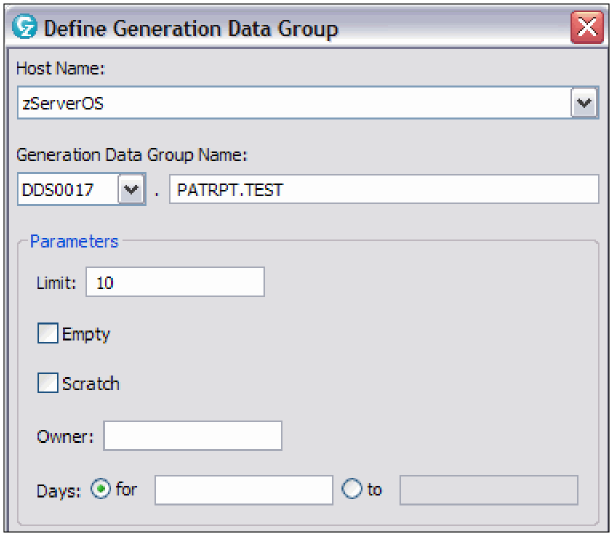


Figure 12-44 Specifying the parameters for the new GDG base

3. **Preparation of JCL**

   a. Modify the Property Group associated with the MVS Subproject "Redbook" with the specific Compile, Link-Edit, and Run libraries and parameters (Figure 12-45).

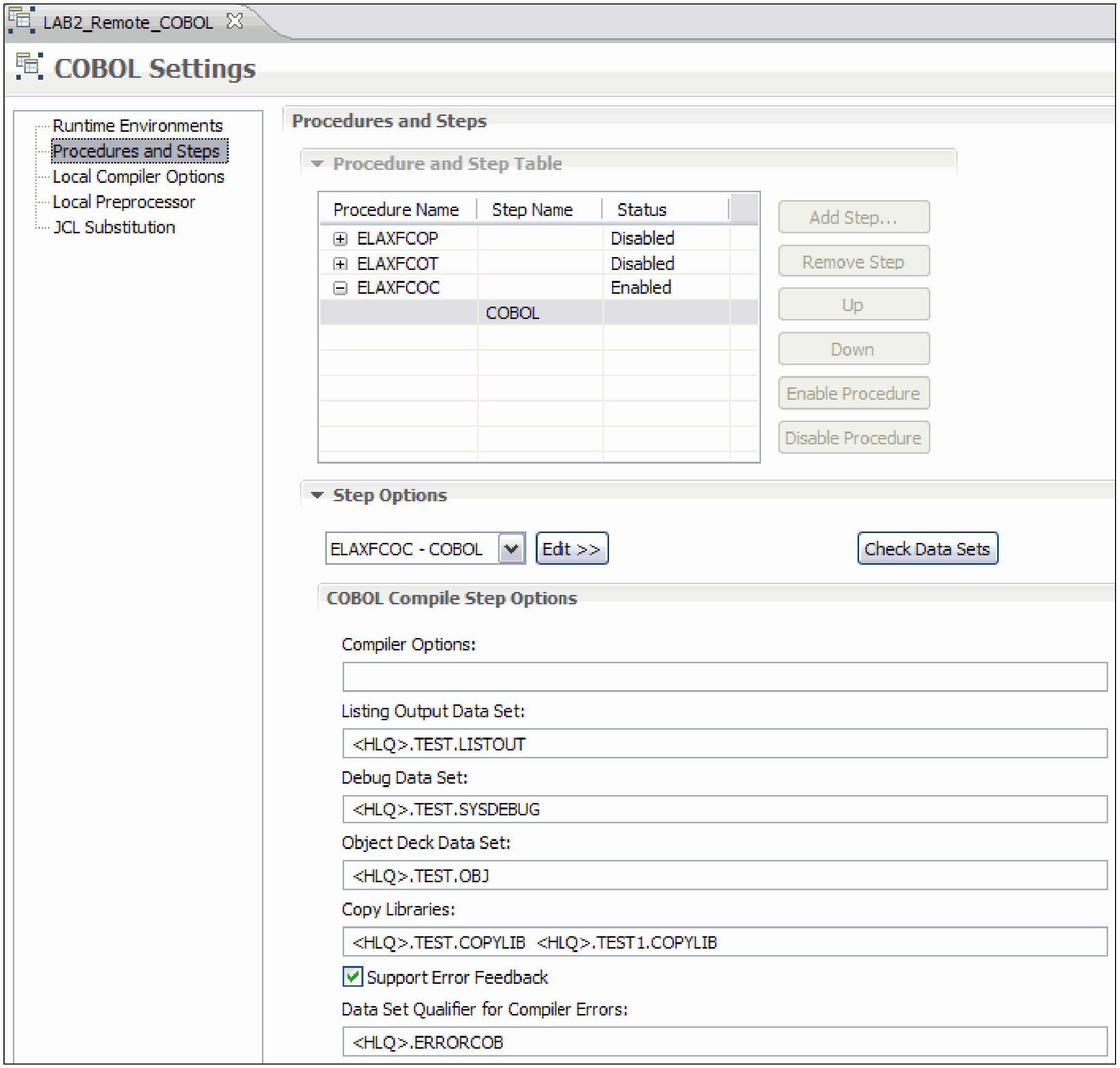


Figure 12-45 Preparing Property Group for the project

b. After the Property Group is associated properly, we generate the JCL, needed to compile, link-edit, and run our modified programs (Figure 12-46 and Figure 12-47).

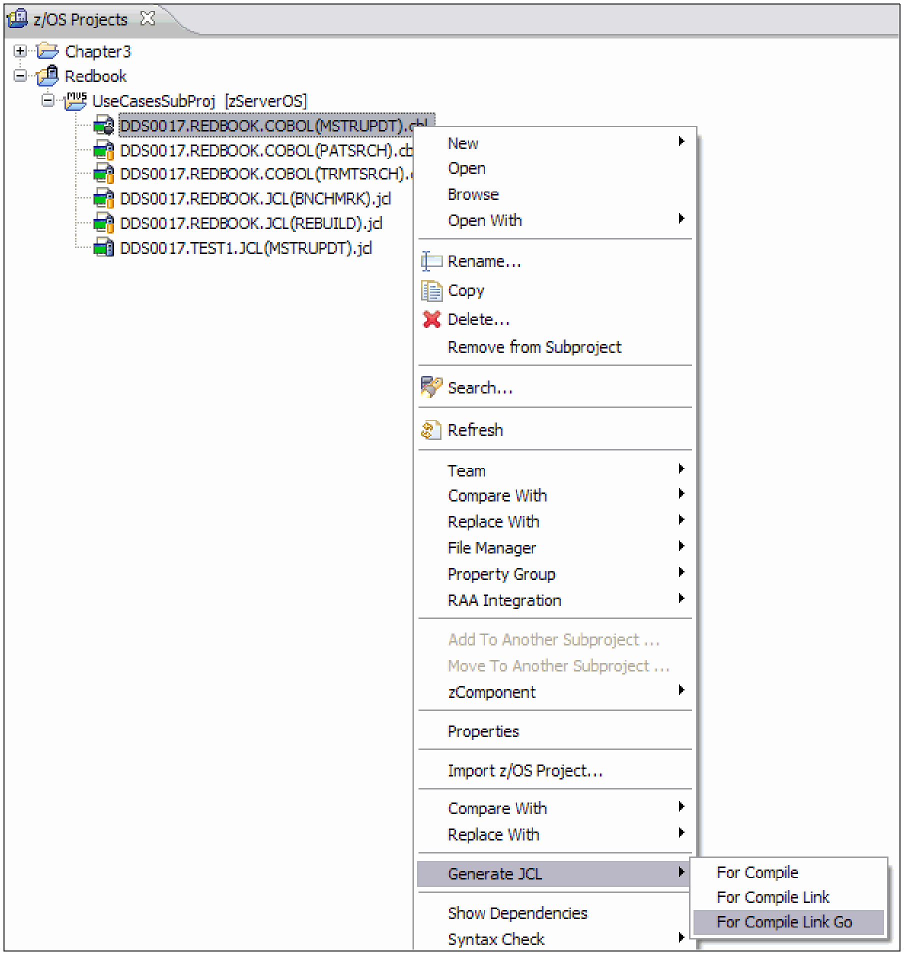


Figure 12-46 Generating JCL for Compile Link Go

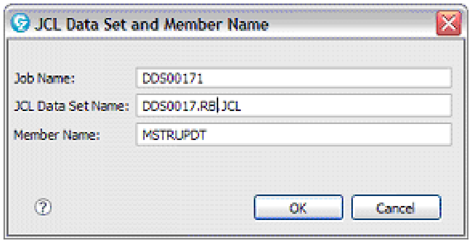


Figure 12-47 Specifying where the generated JCL will be stored

JCL: Preparing JCL, in ISPF, is a manual effort and definitely will require more time than using the automated features in RDz.

4. Running the JCL in debug mode

Before running the aforementioned job generated for our modified code, we generate a new job for the production version of the Program MSTRUPDT and run it against the test I/O files. This will create the PATRPT file for the production version of the program – DDS0017.PATRPT.TEST.G0006V00.

Now we run the JCL MSTRUPDT in debug mode that was created for the modified program. We use debug mode for various reasons such as these:

- We can view the program execution, line-by-line.
- We can verify the value of a variable, during program execution.

- ► We can stop and start program execution, and analyze results at the speed that our procedural understanding of the application's execution flow can handle (Figure 12-48).

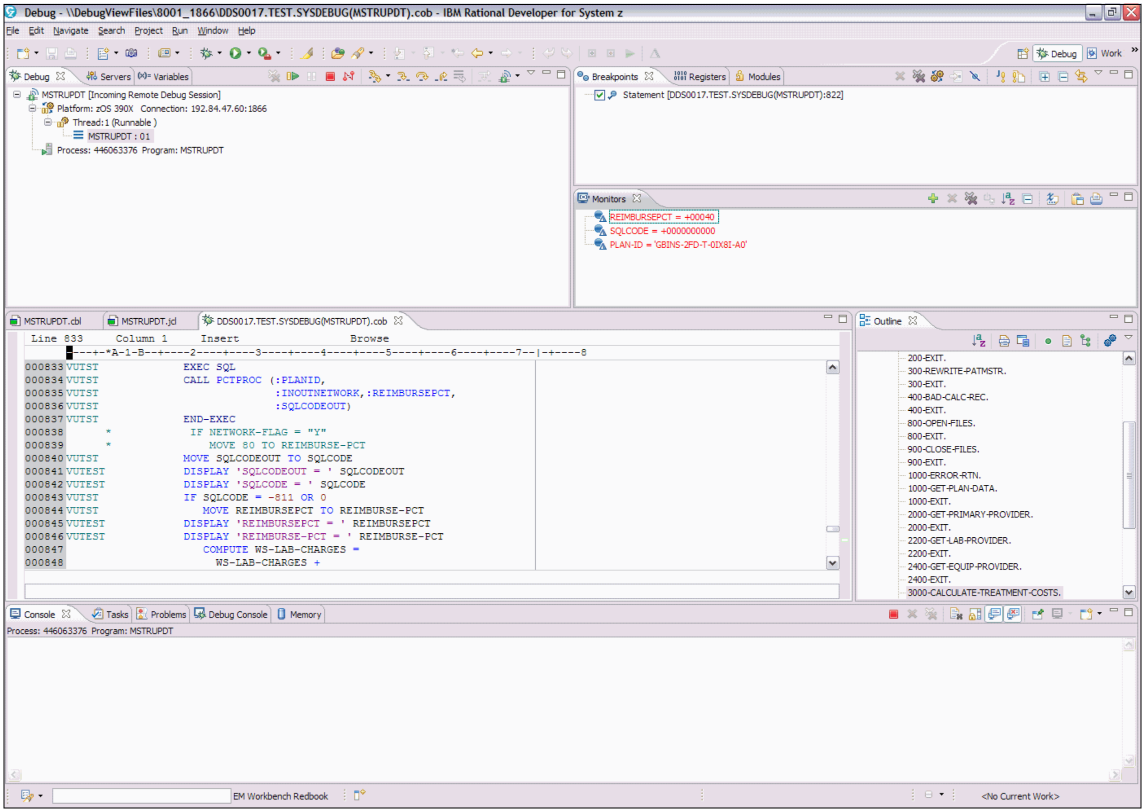
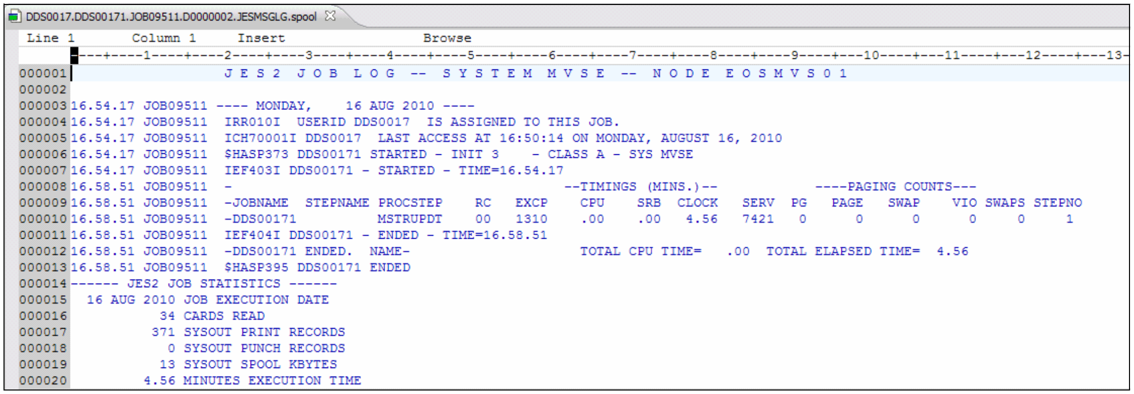


Figure 12-48 Running the JCL in debug mode with various views to help debugging

While debugging, we Step Over the statements that we coded, that is, the call to stored procedure and computation of REIMBURSE-PCT, and test real-time if the code changes are working properly. After this is assured, we resume the job and complete it. Then we check the job spool and the output files to see everything for processed fine (Figure 12-49).



```
DDS0017.DDS00171.JOB09511.D0000002.JESMSGLG.spool
Line 1        Column 1      Insert                 Browse
                   J E S 2  J O B  L O G  --  S Y S T E M  M V S E  --  N O D E  E O S M V S 0 1

16.54.17 JOB09511 ---- MONDAY,    16 AUG 2010 ----
16.54.17 JOB09511  IRR010I  USERID DDS0017  IS ASSIGNED TO THIS JOB.
16.54.17 JOB09511  ICH70001I DDS0017  LAST ACCESS AT 16:50:14 ON MONDAY, AUGUST 16, 2010
16.54.17 JOB09511  $HASP373 DDS00171 STARTED - INIT 3    - CLASS A - SYS MVSE
16.54.17 JOB09511  IEF403I DDS00171 - STARTED - TIME=16.54.17
16.58.51 JOB09511  -                                        --TIMINGS (MINS.)--            ----PAGING COUNTS---
16.58.51 JOB09511  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP    CPU    SRB  CLOCK   SERV  PG   PAGE   SWAP    VIO SWAPS STEPNO
16.58.51 JOB09511  -DDS00171          MSTRUPDT    00   1310    .00    .00   4.56   7421   0      0      0      0     0      1
16.58.51 JOB09511  IEF404I DDS00171 - ENDED - TIME=16.58.51
16.58.51 JOB09511  -DDS00171 ENDED.  NAME-                     TOTAL CPU TIME=   .00  TOTAL ELAPSED TIME=  4.56
16.58.51 JOB09511  $HASP395 DDS00171 ENDED
------ JES2 JOB STATISTICS ------
  16 AUG 2010 JOB EXECUTION DATE
           34 CARDS READ
          371 SYSOUT PRINT RECORDS
            0 SYSOUT PUNCH RECORDS
           13 SYSOUT SPOOL KBYTES
         4.56 MINUTES EXECUTION TIME
```

Figure 12-49   JCL run to completion successfully

The job creates the new generation output file PATRPT file: DDS0017.PATRPT.TEST.G0007V00. We can compare this output file with the one created using the production version and check the differences. We know that the difference must only exist in the TOTAL CHARGES field as this is computed using REIMBURSE-PCT whose logic has been changed as part of the the Change Request (Figure 12-50 and Figure 12-51).

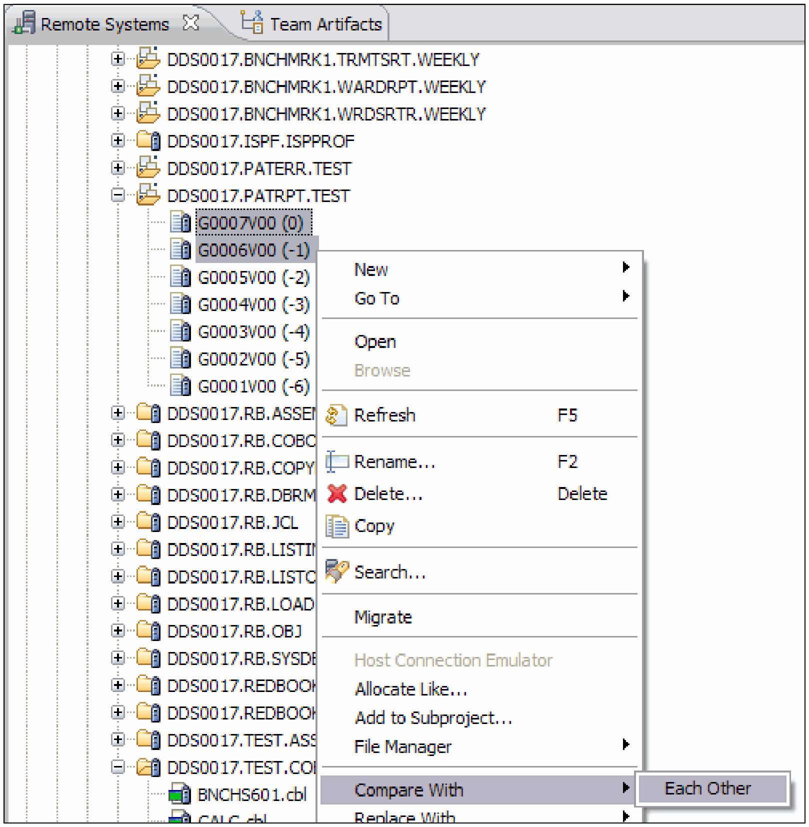


Figure 12-50 Comparing two GDG versions

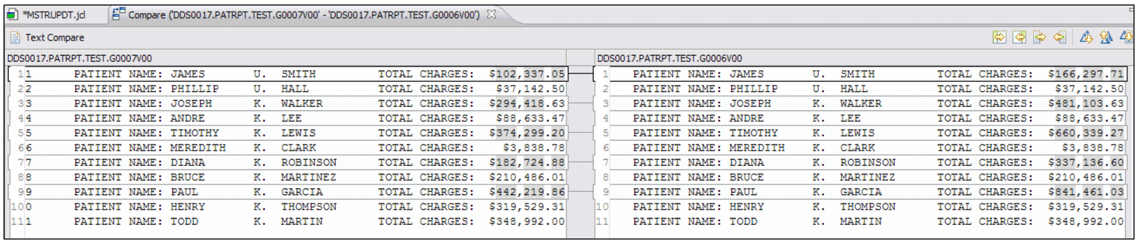


Figure 12-51 Shows comparison results

Similarly, the test cases for the code changes to the programs PATSRCH and TRMTSRCH are performed using the steps as described above.

> **Features:** Debugging a program, using breakpoints, setting watches and monitors, having the option to see different views simultaneously, and various other options, has never been so easy. Using ISPF, it is a very cumbersome task to achieve this.

Regression testing

On ensuring that all the exit-criteria's for the Unit Testing is a PASS, we do the Regressing testing by running the whole Batch Stream according to the Regression Test Plan. All the steps are verified to see if they ran fine and the outputs reports are created according to the business requirements.

IMP:

- All the test case results are reviewed to check if all the exit criteria are satisfied and signed off by the reviewer.
- The signed off components are migrated to the Test environment using the process specified by the clients.

User acceptance testing

In most cases, there is a dedicated team that does the user acceptance testing. They prepare their own test cases, perform them and validate them. On successful completion of the UAT, the developer is notified of the results and sign off is given to go ahead with the installation in production. In case of defects during UAT, they are logged in the tool, specified by the client shop and assigned to the developer for being fixed. The developer performs the check out process, fixes the code and checks the component back in for the team to continue the UAT testing.

13

Non-Trivial Change Request use case

In this section, we document the use of integrated tools in the optimized life cycle to process what we call a "Trivial Change Request". Specifically, we focus on using the tools introduced in previous sections to accomplish the various tasks associated with this kind of request:

- Specifying the business requirements and functional requirements
- Performing a static (and possibly dynamic) analysis of the impact that a proposed change will have on the application – and upstream/downstream processing
- Performing the development activities including coding and testing
- Highlighting the benefits of using the tools over ISPF

13.1 Introduction: Non-Trivial Change Request use case

Change Requests can be defined as a call for modifying the behavior of a system due to normal business changes or because there is a bug in the system. A change request needs to go through a software development cycle in which the impact of the considered solution and effort is analyzed and estimated for that change, and then it has to be approved for implementation before work on it can begin. In short, the goal of the Change Request process is to produce a more robust system that better supports the business activities of our customers.

Based on the complexity and the effort required to complete a change request, we can term them as trivial or non-trivial. Non-Trivial requirements change is one that disrupts and affects overall system performance. This section will show how to use the tools and processes of the optimized life cycle to complete the change request successfully.

13.2 Business requirement

At present, the healthcare business application OnDemandHealth has the LAB TEST ID as of 8 digits. There is a merger and the organization getting merged is having the LAB TEST ID as 12 digits. To keep both the systems in sync, a decision is taken to increase the present LAB TEST ID from 8 to 12.

There is no functional requirement based on the variable length change. LAB-TEST-ID is the key of any treatment transaction. The change requirement is limited to the length of the variable LAB-TEST-ID, cross-reference variables, related variables in the copybook, JCL sorting step, and the change in the length of the report or intermediate file(s) that hold the variables.

The Project Manager or the Project Lead, at this point of time, will open a Work Item in RTCz documenting the approved Change Request and will assign it to the resources required to complete the task (Figure 13-1).

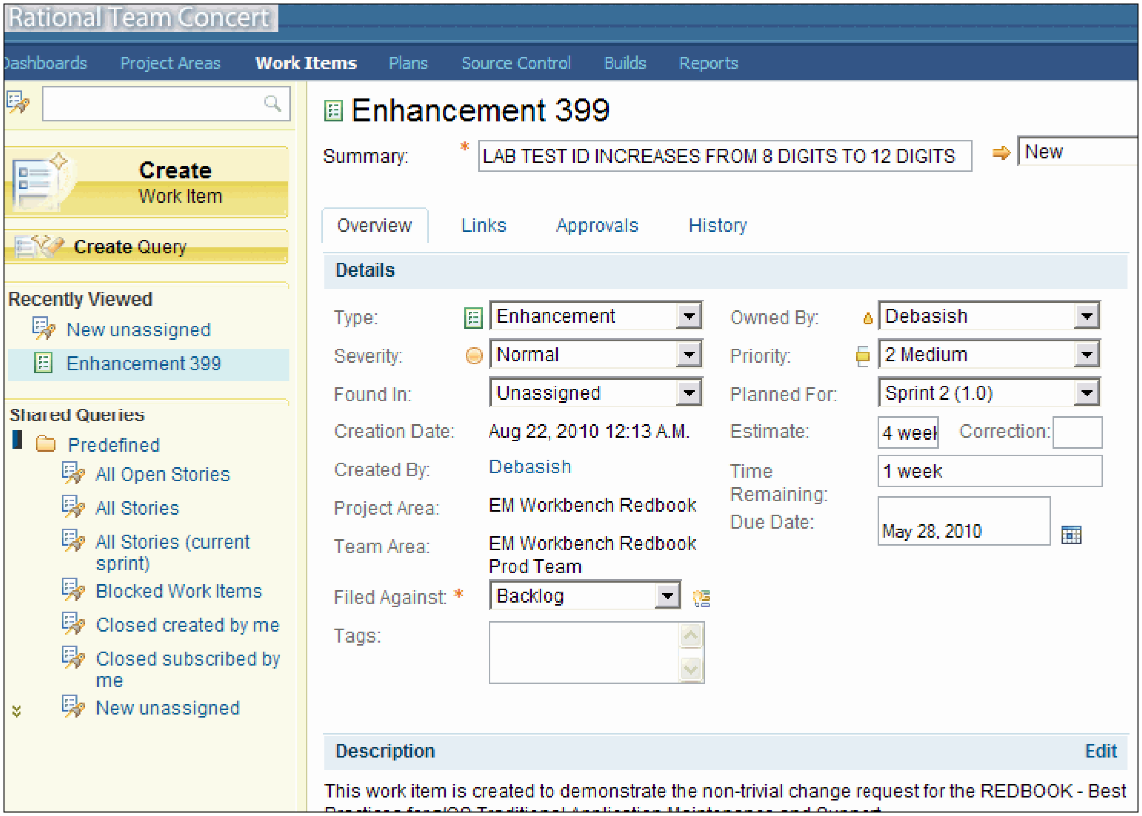


Figure 13-1 RTCz Work Item

13.3 Impact Analysis

Impact Analysis is a very important part of a Change Request, which provides an initial insight to the level of complexity and the amount of effort required to achieve the change. Using RAAi, we perform an impact analysis on the variable REIMBURSE-PCT, because this is the data-element whose computational logic is going to be modified.

Impact analysis results shows that there are three Source programs and one JCL impacted, apart from a number of data elements, which need to be further analyzed.

Impacted JCL, COBOL programs, and copybook

The impacted JCL, COBOL programs, and copybook are shown in Figure 13-2.

File (9)	Langu age	Type	Analysis status	No# of lines	Source location
BNCHMRK.jcl	JCL	Batch job source	Completed	306	C:/D-Drive/RDZProjectNew/DemoProj/JCL/BNCHMRK.jcl
MSTRUPDT.cbl	COBOL	Program source	Completed	638	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/MSTRUPDT.cbl
PATLIST.cbl	COBOL	Program source	Completed	962	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/PATLIST.cbl
TREATMNT.cpy	COPY BOOK	Included source	Completed	64	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/TREATMNT.cpy
TRMTSRCH.cbl	COBOL	Program source	Completed	422	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/TRMTSRCH.cbl
TRMTUPDT.cbl	COBOL	Program source	Completed	430	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/TRMTUPDT.cbl
TRTMNT.cbl	COBOL	Program source	Completed	794	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/TRTMNT.cbl

Figure 13-2 Impact Analysis on RAAi

Impacted files and datasets

The impacted files and datasets are shown in Figure 13-3.

Data store (10)	Type	Program	Source location
LABTEST	FILE	TRMTSRCH	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/TRMTSRCH.cbl
PATRPT	FILE	MSTRUPDT	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/MSTRUPDT.cbl
PATRPT	FILE	PATLIST	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/PATLIST.cbl
TRMTDATA	FILE	TRTMNT	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/TRTMNT.cbl
TRMTEDIT	FILE	TRTMNT	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/TRTMNT.cbl
TRMTERR	FILE	TRTMNT	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/TRTMNT.cbl
TRMTSORT	FILE	TRMTSRCH	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/TRMTSRCH.cbl
TRMTSRCH	FILE	TRMTUPDT	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/TRMTUPDT.cbl
TRMTSRCH	FILE	TRMTSRCH	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/TRMTSRCH.cbl
TRMTSRCH	FILE	PATLIST	C:/D-Drive/RDZProjectNew/DemoProj/BatchCobol/PATLIST.cbl

Figure 13-3 Impact Analysis on RAAi

Apart from these, there is also a detailed analysis result showing the impacted variables in different programs (due to the bigger size not included in the present scope).

From the RAAi Impact Analysis results, we get a detailed list of all the impacted components in the application. This will help in determining the scope of the Change Request in terms of schedule and effort.

Results: Clearly, in ISPF it is not possible to get this kind of impact analysis results. In ISPF we are manually searching the variable in all the components and deducing the results.

13.4 Change design

The RAA impact analysis results from Figure 13-3 are taken as input and further analysis is performed to prepare a detailed technical design. Some of the critical points of the design for the above change are discussed here – Note that these next few sections are very detailed and describe procedurally oriented tasks at the software practitioner level.

The detailed RAA analysis report is used to mark the reference and impacted fields due to the field (LAB-TEST-ID) change. Here a design document is to be prepared with the impacted elements (Job / Program / Copybook / Data Sets) and necessary changes.

Using RAAi Impact Analysis, we had identified which source modules that need modification. Further to impact analysis, we use the RAAi Integration Search utility in RDz to find further where exactly the changes will take place. The RAAi Search will be performed against LAB-TEST-ID twice. First using the option ‘Limit To’ as Declarations, which indicate the variable declaration in different components. Secondly, with the ‘Limit To’ option as Reference and Modification’, which cover the overall scope o the variable (Figure 13-4 and Figure 13-5.)

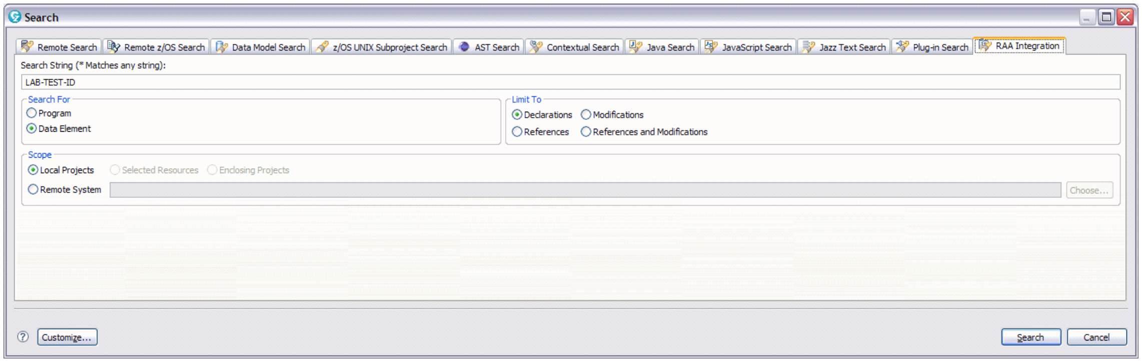


Figure 13-4 RAAi Search on LAB_TEST-ID

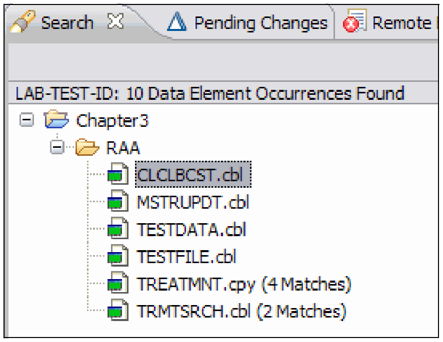


Figure 13-5 RAAi Search results

13.4.1 Modification to the source modules

Using the RAAi results, we identify the following changes to the source modules:

1. **TRTMNT COBOL program**

   Variables INPATIENT-TREATMENT-REC-DATA, INPATIENT-TREATMENT-REC-EDIT, and REST-OF-REC must be increased by 4 bytes.

2. **TRMTSRCH COBOL program**

   Variables INPATIENT-TREATMENT-REC-SORT, INPATIENT-TREATMENT-REC-SRCH, LAB-TEST-ID must be increased by 4 bytes.

3. **TRMTUPDT COBOL program**

   Variables INPATIENT-TREATMENT-REC-SRCH, HOLD-LAB-TEST-ID.

4. **MSTRUPDT COBOL program**

   Variables PATMSTR-REC, LAB-TEST-ID must be increased by 4 bytes

5. **PATLIST COBOL program**

   Variables INPATIENT-TREATMENT-REC-SRCH, RPT-REC, and PATLISTEST-ID-O must be increased by 4 bytes.

6. **TREATMNT copybook**

   LAB-TEST-ID must be increased by 4 bytes

13.4.2 Modifications to JCL

From the RAAi Impact Analysis results and as a result of the changes to the source modules, the following changes will happen to the different steps in the JCL BNCHMRKQ:

1. **TRMTEDIT**

   LRECL of RTPOT06.BNCHMRK.TRMTEDIT.WEEKLY and RTPOT06.BNCHMRK.TRMEDERR.WEEKLY GDGs will change from 1101 to 1105.

2. **TRMTSORT**

   LRECL of RTPOT06.BNCHMRK.TRMTSORT.WEEKLY GDG will change from 1101 to 1105.

3. **TRMTSRCH**

   LRECL of RTPOT06.BNCHMRK.TRMTSRCH.WEEKLY GDG is to be changed from 1101 to 1105

4. **PATRPT**

   LRECL of RTPOT06.BNCHMRK.PATRPT.WEEKLY GDG is to be changed from 133 to 137

13.4.3 Test plan

In the Design phase itself, we prepare the Test Plan (for any change – including trivial changes), in parallel, based on the design. The test plan ideally contains the entry and exit criteria for the change being implemented. RAAi can also help with the test plan – as it surfaces file and database dependencies for a program.

A sample test case is shown in Table 13-1.

Table 13-1 Table 16-1 Sample Test Case

Sl No	Action Involved	File/Database Table	Expected Result	Actual Result	Pass/Fail
1	After the necessary changes with compilation, link-bind, run the job BNCHMRKQ	PATRPT file	The new PATRPT (RTPOT06.BNCHMRK.PATRPT.WEEKLY) report must be same as of the old PATRPT (RTPOT06.BNCHMRK.PATRPT.WEEKLY) report except the change of length of the field LAB-TEST-ID		

IMP:

– The Design and Test Plan must undergo the review as per the Quality process.

– The Detailed Technical Design Document must be signed-off by client.

– The Test Case document must be signed-off by client.

13.5 Development

After all the necessary signoffs are received, the development of the change is started. Development phase involves coding the changes and then testing them till the exit criteria is satisfied.

13.5.1 Coding

In this section we discuss considerations regarding the coding.

Code modification to the existing modules

RTCz is used here to do the source control management. After connecting to the Repository and creating a Repository Workspace, we follow the below steps to retrieve the components from Production to do the code modifications:

- Create an MVS Subproject in RDz, where the checked-out components will be retrieved to (Figure 13-6).

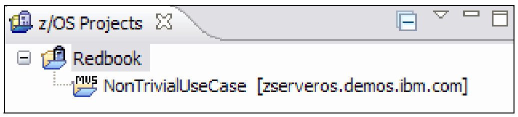


Figure 13-6 MVS Subproject created

- Switch to the Work Items Perspective and check-out the components to be modified using the ‘Loading zFiles to zOS…’ feature of RTCz (Figure 13-7).

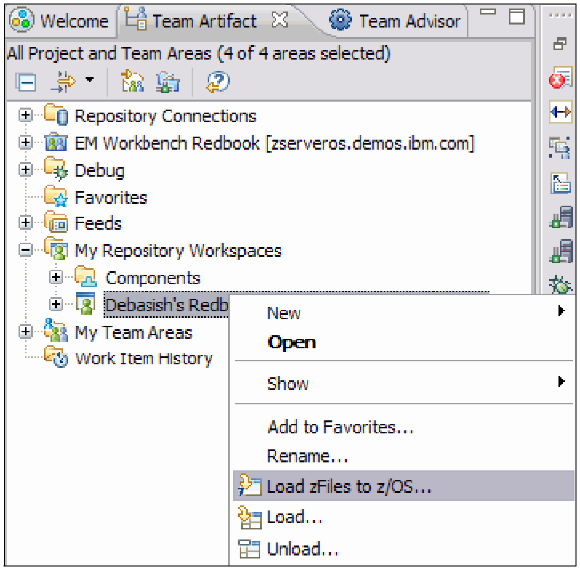


Figure 13-7 Option to Check-Out the components in RTCz

- The components TREATMENT.cpy, MSTRUPDT.cbl, PATLIST.cbl, TRTMNT.cbl, TRMTUPDT.cbl, TRMTSRCH.cbl, and BNCHMRK.jcl are selected (Figure 13-8).

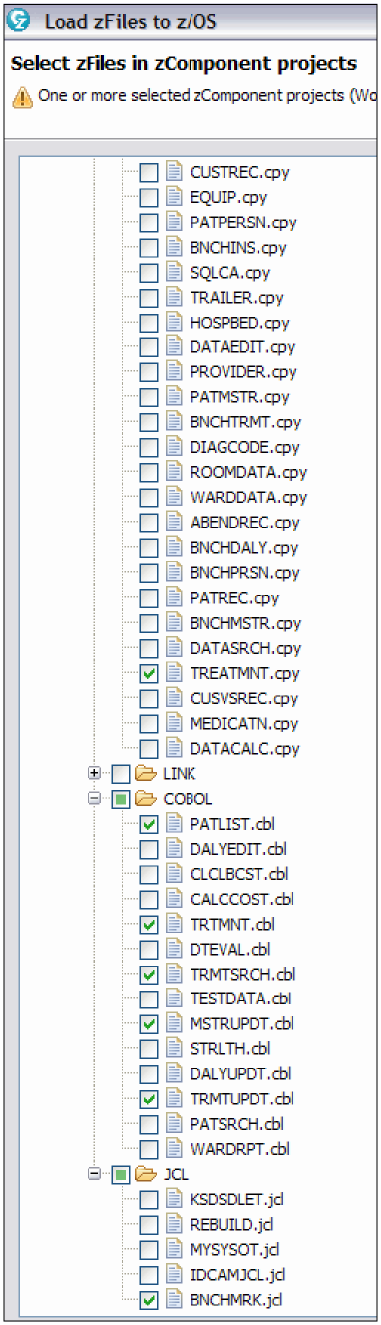


Figure 13-8 Selecting components to be checked out

After the steps in the wizard are completed, the components are checked out to the MVS Subproject specified (Figure 13-9).

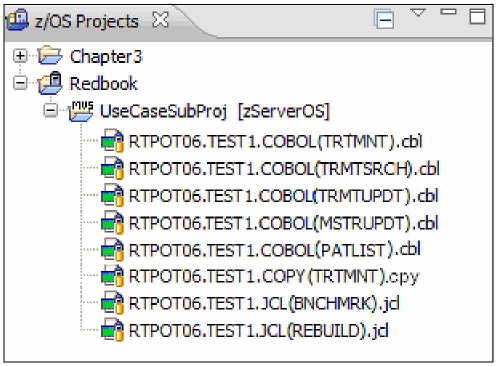


Figure 13-9 Components checked out into the MVS Subproject

MSTRUPDT COBOL program

The checked out TRMNT program is opened in edit mode from the UseCaseSubProj. The variables PATMSTR-REC and LAB-TEST-ID is increased by 4 bytes as shown in the and saved (Figure 13-10).

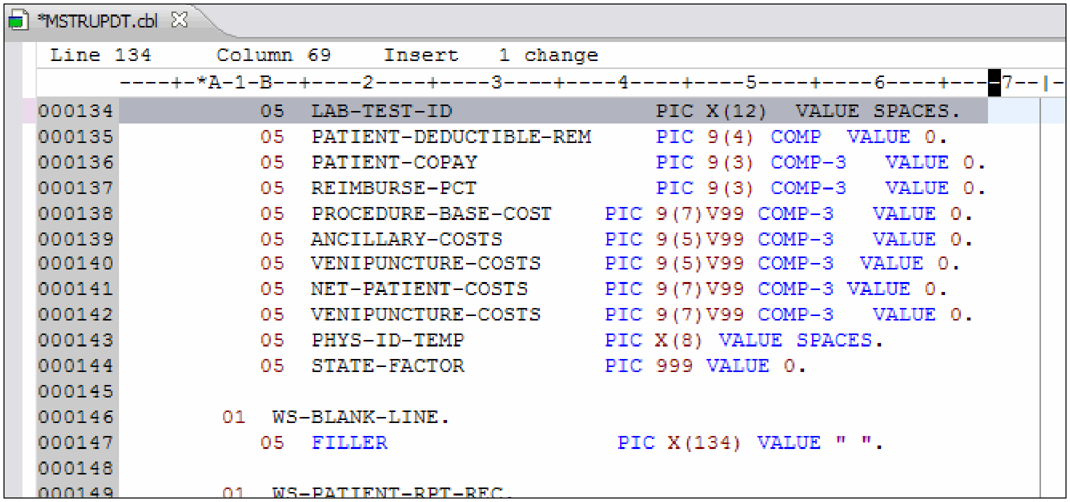


Figure 13-10 Coding the changes in MSTRUPDT

Similarly, we do the code changes for the COBOL programs TRTMNT, TRMTSRCH, TRMTUPDT, PATLIST, copybook TREATMENT and the JCL BNCHMRKQ as per the Design document prepared earlier in the section.

On completing the coding and the required reviews, the modified components are checked in to the repository workspace in RTCz using the 'Pending Changes' view (Figure 13-11).

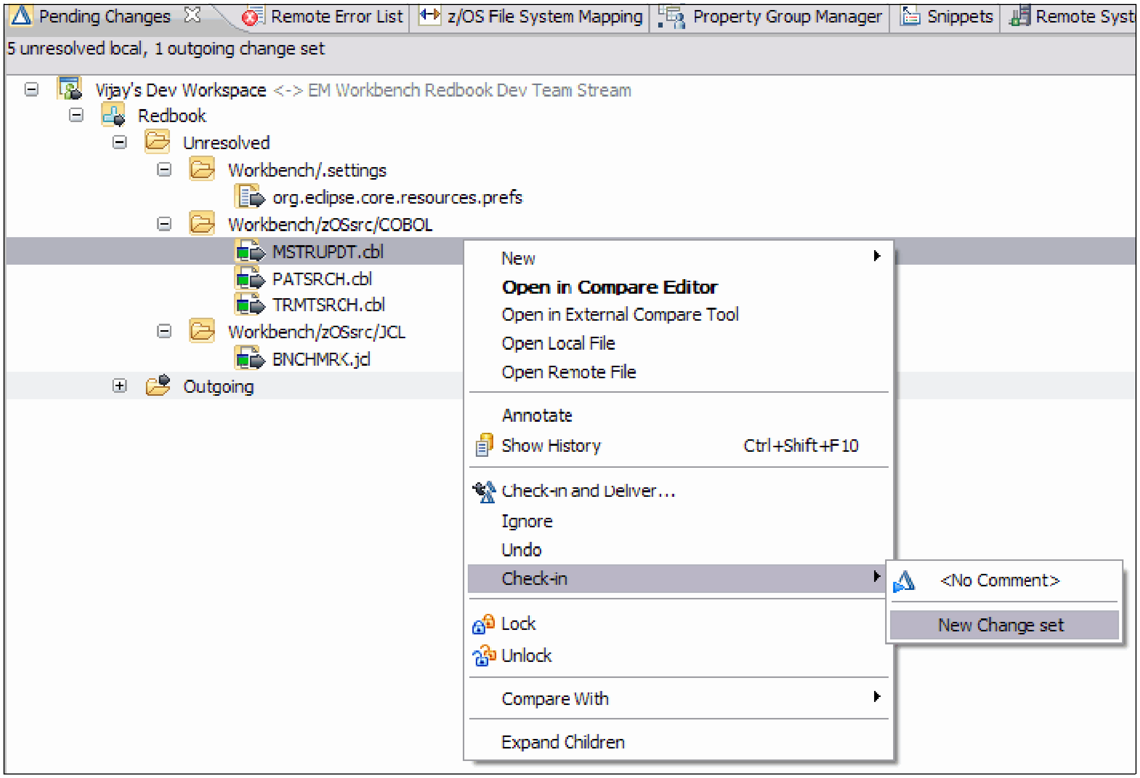


Figure 13-11 Code changes in MSTRUPDT being checked in

> **RDz:** It is very evident here that the numerous features available in RDz, makes coding a lot more easier and convenient task compared to ISPF. In ISPF, we do not have the options such as real-time syntax checks, Content Assist, commenting a bulk of statements.

IMP:

- Update the revision history of the changed module and tag the code changes accordingly as per coding standards specified by the client
- All the code changes must be properly reviewed as per the Quality process

13.5.2 Testing

Testing is done with the help of the PD Tools integrated with RDz and Data Perspective, as per the Test Plan prepared during Design phase.

Unit Testing

In this section we discuss various considerations involving Unit Testing.

1. Preparation of Test Data - PATMSTR

- Using the RAAi Program Diagram, we find out what data are input to the program and what the output from the program for testing (Figure 13-12).

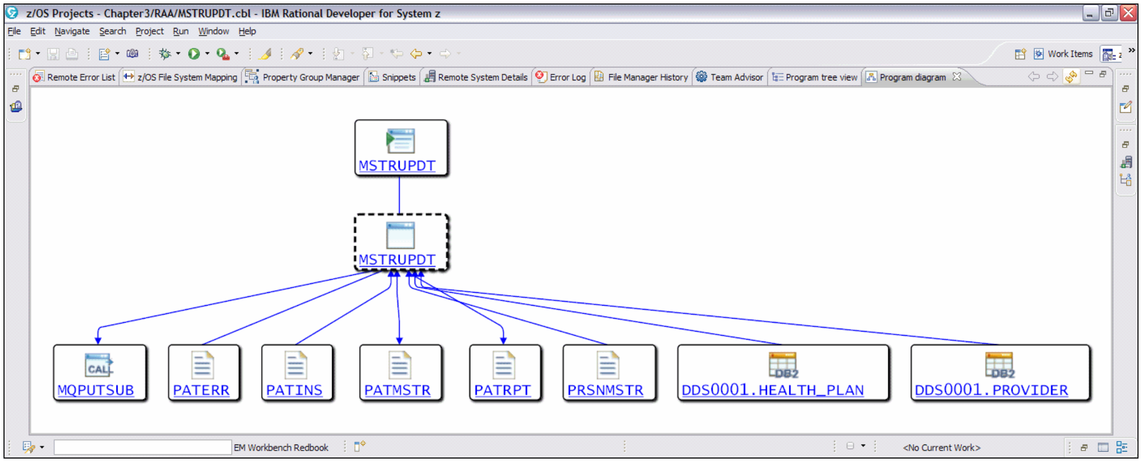


Figure 13-12 RAAi Program Diagram view

- Using the File Manager features integrated with RDz, the input data is prepared by allocating the VSAM files and copying test data from the production files:
  - Allocate a new VSAM file TEST.PATMASTR with the parameters such as the existing PATMASTR production VSAM file but the length of the file is changed from 2964 to 2968 (Figure 13-13, Figure 13-14, and Figure 13-15).

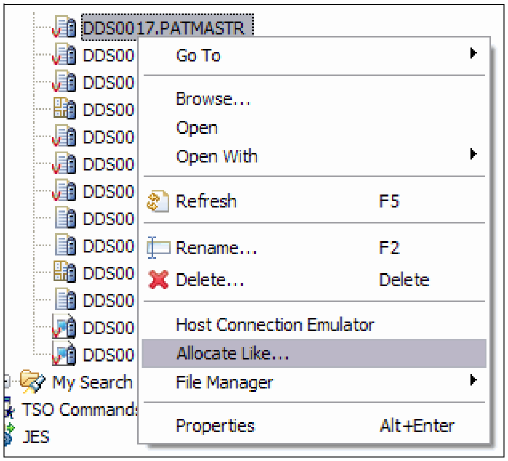


Figure 13-13 Using Allocate Like... option to allocate VSAM dataset

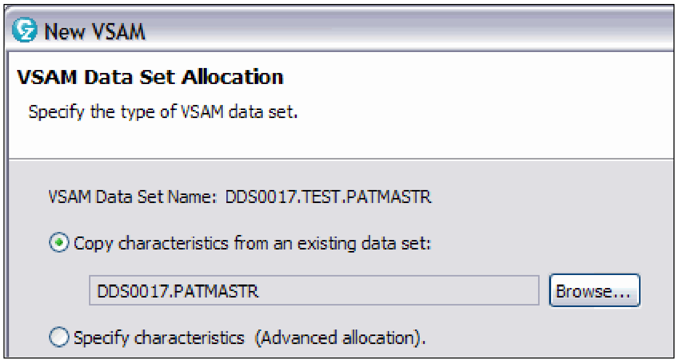


Figure 13-14 Using the parameters of an existing VSAM dataset

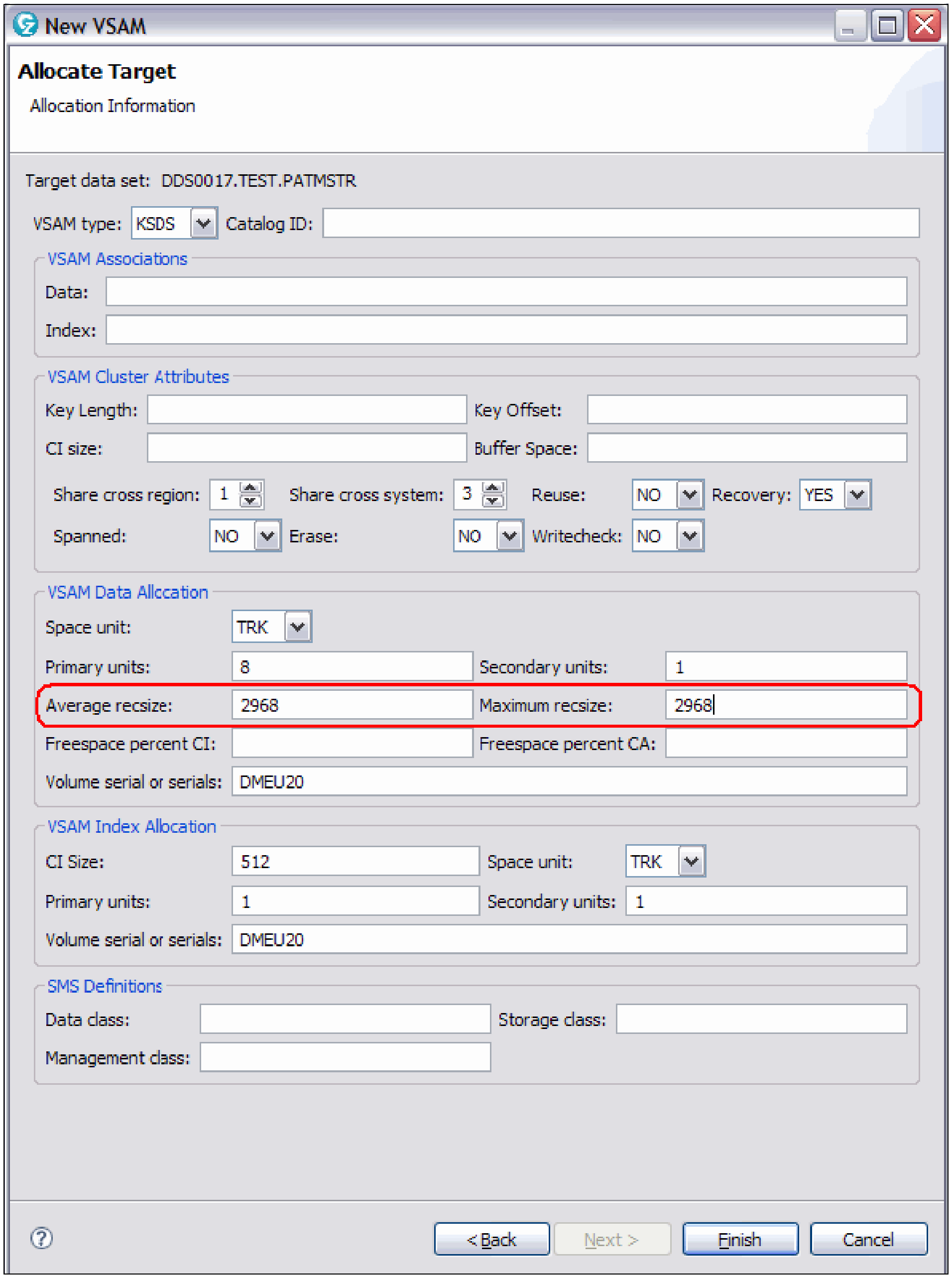


Figure 13-15 Modify any parameter for the target dataset, if required

– After the VSAM file is created, we copy sample data from the production file (Figure 13-16, Figure 13-17, and Figure 13-18).

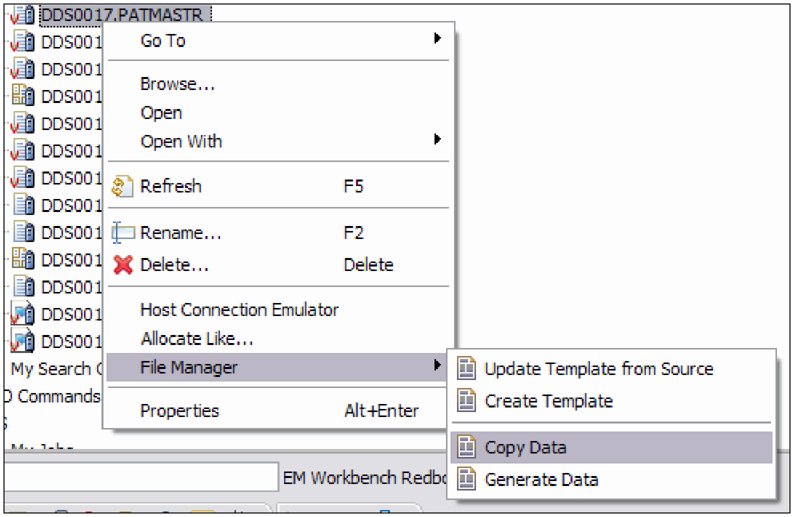


Figure 13-16 Copying data to the new VSAM dataset from an existing one

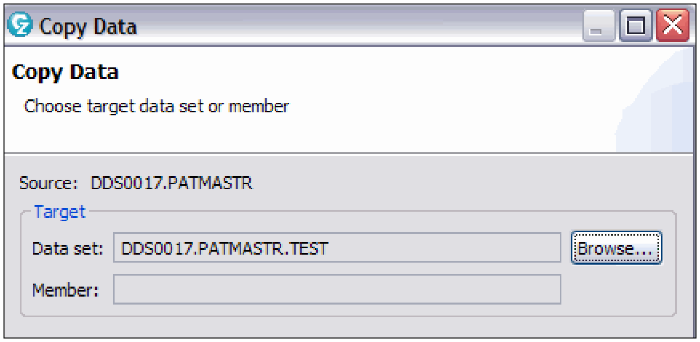


Figure 13-17 Selecting the existing dataset to be copied

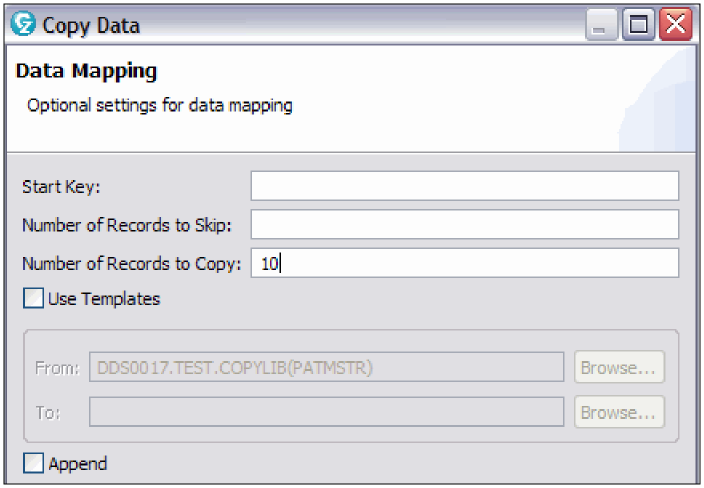


Figure 13-18 Specifying numbers of records to be copied

– Associate the template with the file created and modify the data, if required, as per our test cases and save it (Figure 13-19 and Figure 13-20).

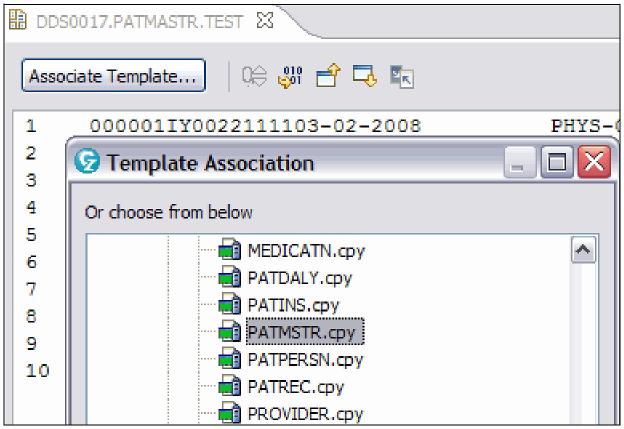
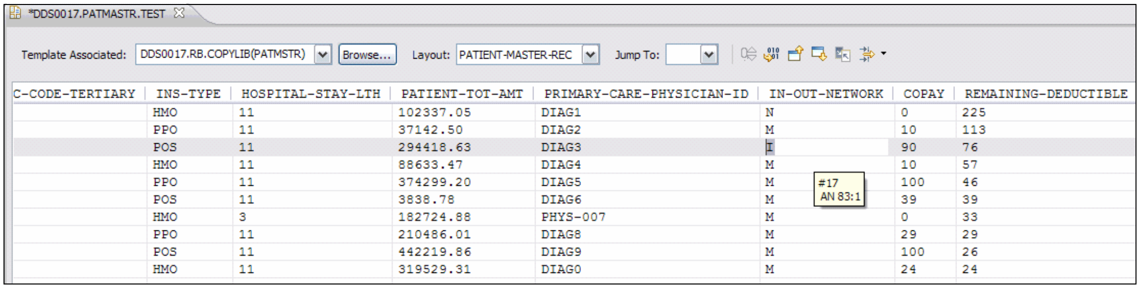


Figure 13-19 Selecting the Template to be used



C-CODE-TERTIARY	INS-TYPE	HOSPITAL-STAY-LTH	PATIENT-TOT-AMT	PRIMARY-CARE-PHYSICIAN-ID	IN-OUT-NETWORK	COPAY	REMAINING-DEDUCTIBLE
	HMO	11	102337.05	DIAG1	N	0	225
	PPO	11	37142.50	DIAG2	M	10	113
	POS	11	294418.63	DIAG3	I	90	76
	HMO	11	88633.47	DIAG4	M	10	57
	PPO	11	374299.20	DIAG5	M	100	46
	POS	11	3838.78	DIAG6	M	39	39
	HMO	3	182724.88	PHYS-007	M	0	33
	PPO	11	210486.01	DIAG8	M	29	29
	POS	11	442219.86	DIAG9	M	100	26
	HMO	11	319529.31	DIAG0	M	24	24

Figure 13-20 New VSAM dataset created

2. **Test Input Data file creation – TRMTDATA**

   The production file TRMTDATA (one-day transaction from online system) is copied from the production environment to the test environment. Test File copied from Production has a record length of 1101. And LAB-TEST-ID is positioned from 290 to 298 (Figure 13-21).

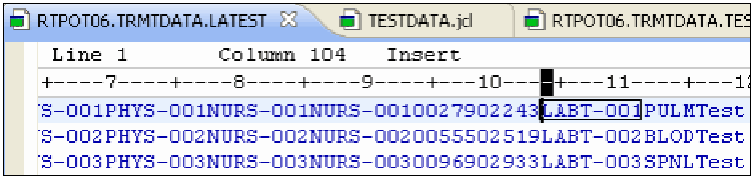


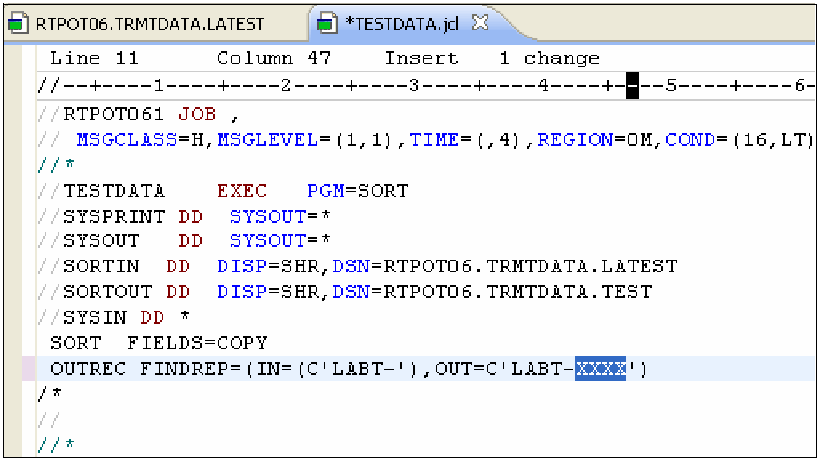
Figure 13-21 Unit Test Data file copied from production

3. Test Input Data file modification for LAB-TEST-ID field

In the normal job flow, the input file for the batch cycle comes from the online screen. When the field LAB-TEST-ID in the online screen increases by 4 bytes, again the TRMTDATA file comes with the extended data.

For the time being, when the online modified program is out of scope, to test the modified program, the LAB-TEST-ID field of the input file needs to be prepared with data from 8 bytes to 12 bytes.

Using the DFSORT the field LAB-TEST-ID in the I/P file is increased from 8 to 12 characters (Figure 13-22).



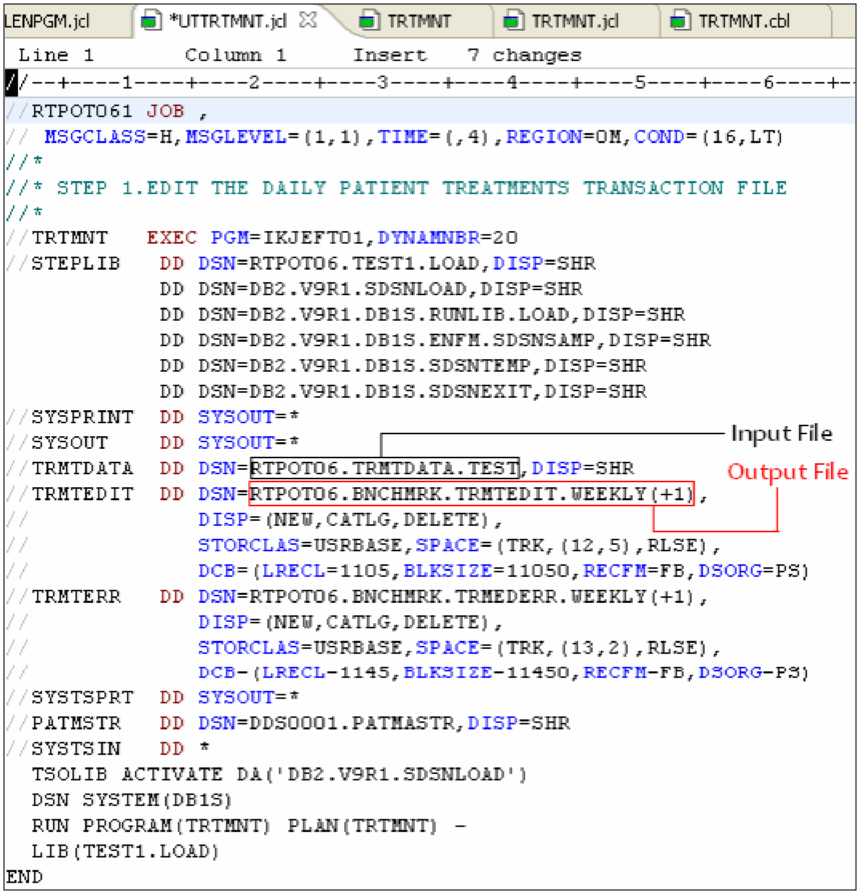
```
RTPOT06.TRMTDATA.LATEST    *TESTDATA.jcl
 Line 11       Column 47     Insert    1 change
//--+----1----+----2----+----3----+----4----+----5----+----6-
//RTPOT061 JOB ,
// MSGCLASS=H,MSGLEVEL=(1,1),TIME=(,4),REGION=0M,COND=(16,LT)
//*
//TESTDATA    EXEC   PGM=SORT
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//SORTIN  DD  DISP=SHR,DSN=RTPOT06.TRMTDATA.LATEST
//SORTOUT DD  DISP=SHR,DSN=RTPOT06.TRMTDATA.TEST
//SYSIN DD *
 SORT  FIELDS=COPY
 OUTREC FINDREP=(IN=(C'LABT-'),OUT=C'LABT-XXXX')
/*
//
//*
```

Figure 13-22 Modification of Unit Test Data file

Now the updated input file is of 1105 record-length and the field LAB-TEST-ID is of 12 character lengths. The modified TESTDATA file RTPOT06.TRMTDATA.TEST is fed to the Unit Test as shown in Figure 13-22.

4. Preparation of Unit Test JCL

The Unit Test JCL is prepared with the new input and output files as shown in Figure 13-23.




```
 Line 1        Column 1      Insert    7 changes
//--+----1----+----2----+----3----+----4----+----5----+----6----+-
//RTPOT061 JOB ,
//  MSGCLASS=H,MSGLEVEL=(1,1),TIME=(,4),REGION=0M,COND=(16,LT)
//*
//* STEP 1.EDIT THE DAILY PATIENT TREATMENTS TRANSACTION FILE
//*
//TRTMNT    EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB    DD DSN=RTPOT06.TEST1.LOAD,DISP=SHR
             DD DSN=DB2.V9R1.SDSNLOAD,DISP=SHR
             DD DSN=DB2.V9R1.DB1S.RUNLIB.LOAD,DISP=SHR
             DD DSN=DB2.V9R1.DB1S.ENFM.SDSNSAMP,DISP=SHR
             DD DSN=DB2.V9R1.DB1S.SDSNTEMP,DISP=SHR
             DD DSN=DB2.V9R1.DB1S.SDSNEXIT,DISP=SHR
//SYSPRINT   DD SYSOUT=*
//SYSOUT     DD SYSOUT=*
//TRMTDATA   DD DSN=RTPOT06.TRMTDATA.TEST,DISP=SHR
//TRMTEDIT   DD DSN=RTPOT06.BNCHMRK.TRMTEDIT.WEEKLY(+1),
//              DISP=(NEW,CATLG,DELETE),
//              STORCLAS=USRBASE,SPACE=(TRK,(12,5),RLSE),
//              DCB=(LRECL=1105,BLKSIZE=11050,RECFM=FB,DSORG=PS)
//TRMTERR    DD DSN=RTPOT06.BNCHMRK.TRMEDERR.WEEKLY(+1),
//              DISP=(NEW,CATLG,DELETE),
//              STORCLAS=USRBASE,SPACE=(TRK,(13,2),RLSE),
//              DCB-(LRECL-1145,BLKSIZE-11450,RECFM-FB,DSORG-PS)
//SYSTSPRT   DD SYSOUT=*
//PATMSTR    DD DSN=DDS0001.PATMASTR,DISP=SHR
//SYSTSIN    DD *
  TSOLIB ACTIVATE DA('DB2.V9R1.SDSNLOAD')
  DSN SYSTEM(DB1S)
  RUN PROGRAM(TRTMNT) PLAN(TRTMNT) -
  LIB(TEST1.LOAD)
END
```

Figure 13-23 Unit Test JCL for TRTMNT program

5. Executing the Unit Test

The Test JCL prepared is submitted and the expected output here is that the output file TRMTEDIT given in the job step must have the LAB-TEST-ID as of 12 bytes. We find that the actual output is as expected as shown in Figure 13-24.

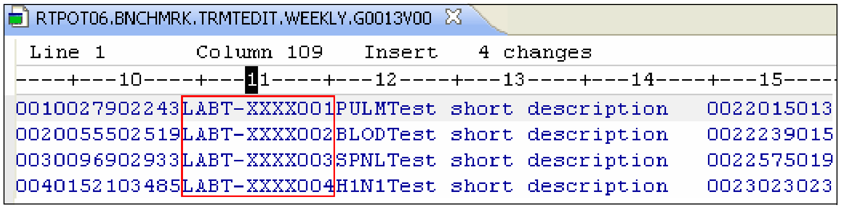


Figure 13-24 Unit Test result

6. Comparing the output results

Through the File compare utility in RDz, both the outputs (Pre-run and Post-run) can be compared to ensure that the changes are limited to the LAB-TEST-ID only, as shown in Figure 13-25 and Figure 13-26.

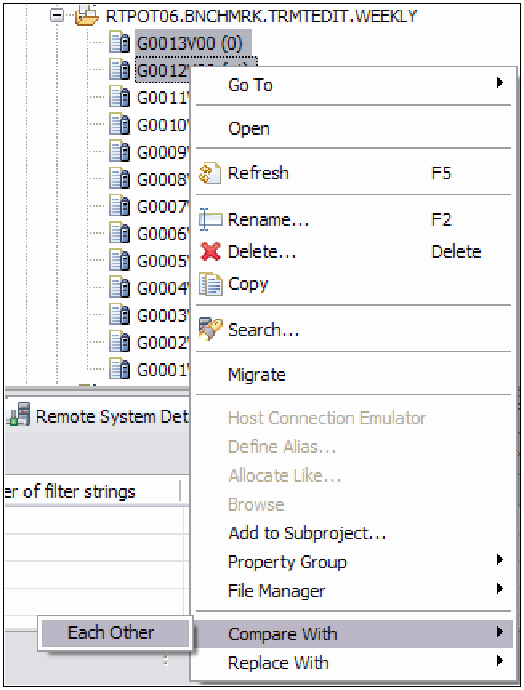


Figure 13-25 Using the Compare utility in RDz

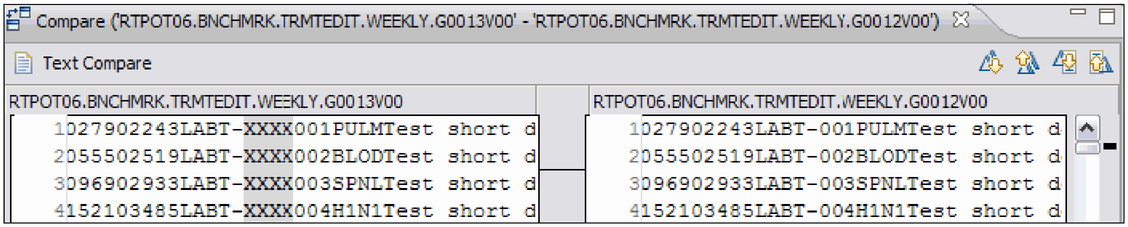


Figure 13-26 The Compare results

Regression testing

To do the regression testing the whole environment needs to be simulated. Both the modified programs and non-impacted programs must be positioned in the testing datasets. Then the batch job BNCHMRK.JCL is executed step by step to create the PATIENT DETAILED LISTING REPORT i.e. till PATRPT step.

REBUILD.jcl is executed excluding the impacted programs to create the load modules for the referred programs

Regression Test is driven by the Regression Test Case created during the design phase to ensure the batch stream works fine with the element level changes. The test run is planned in Testing Environment simulating the sequence of job processing and the data.

Report before the field change:

A report before the field change is shown in Figure 13-27.

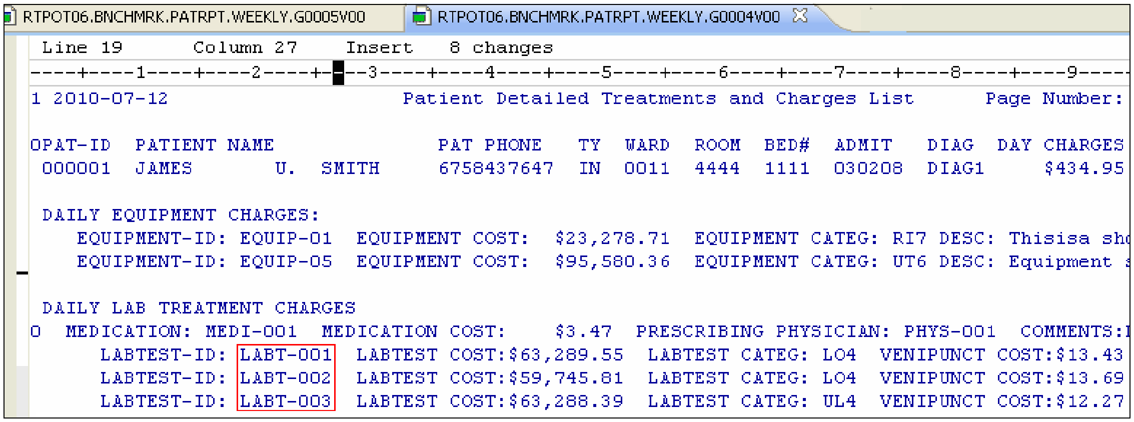
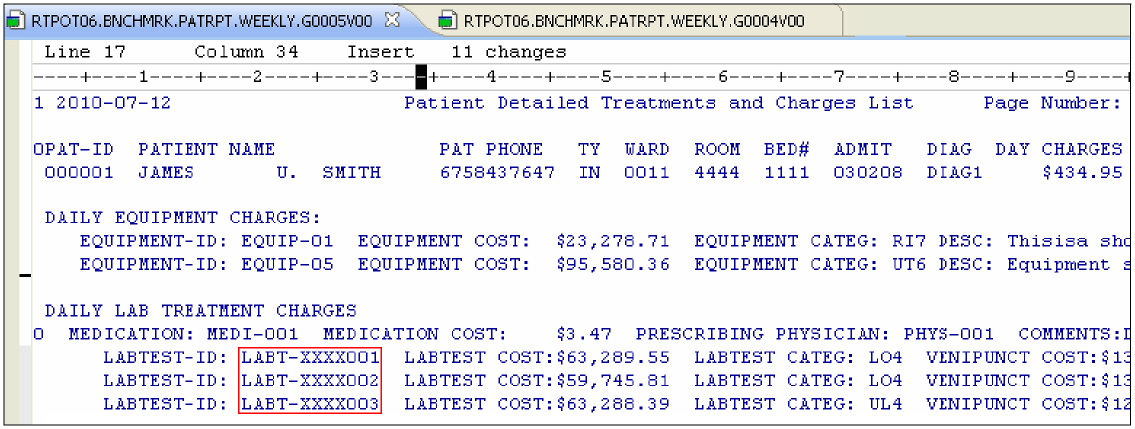


Figure 13-27 Batch Cycle Report before change of the LAB-TEST-ID

Report after the field change:

A report after the field change is shown in Figure 13-28.



```
RTPOT06.BNCHMRK.PATRPT.WEEKLY.G0005V00    RTPOT06.BNCHMRK.PATRPT.WEEKLY.G0004V00
 Line 17       Column 34     Insert     11 changes
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9----+
1 2010-07-12                        Patient Detailed Treatments and Charges List      Page Number:

OPAT-ID  PATIENT NAME               PAT PHONE   TY  WARD  ROOM  BED#  ADMIT   DIAG  DAY CHARGES
 000001  JAMES       U.  SMITH      6758437647  IN  0011  4444  1111  030208  DIAG1      $434.95

 DAILY EQUIPMENT CHARGES:
    EQUIPMENT-ID: EQUIP-01  EQUIPMENT COST:  $23,278.71  EQUIPMENT CATEG: RI7 DESC: Thisisa sho
    EQUIPMENT-ID: EQUIP-05  EQUIPMENT COST:  $95,580.36  EQUIPMENT CATEG: UT6 DESC: Equipment s

 DAILY LAB TREATMENT CHARGES
0  MEDICATION: MEDI-001  MEDICATION COST:    $3.47  PRESCRIBING PHYSICIAN: PHYS-001  COMMENTS:I
      LABTEST-ID: LABT-XXXX001  LABTEST COST:$63,289.55  LABTEST CATEG: LO4  VENIPUNCT COST:$13
      LABTEST-ID: LABT-XXXX002  LABTEST COST:$59,745.81  LABTEST CATEG: LO4  VENIPUNCT COST:$13
      LABTEST-ID: LABT-XXXX003  LABTEST COST:$63,288.39  LABTEST CATEG: UL4  VENIPUNCT COST:$12
```

Figure 13-28 Batch Cycle Report after change of the LAB-TEST-ID

The only change in the report is the size of the LAB-TEST-ID and nothing else. The test case is passed and the regression test is successful. Similar to the Unit Test step, using the file comparison utility both the outputs (Pre-run and Post-run) are compared and the changes are limited to the LAB-TEST-ID only

IMP:

- All the test case results are reviewed to check if all the exit criteria are satisfied and signed off by the reviewer.
- The signed off components are migrated to the Test environment using the process specified by the clients

User acceptance testing

In most cases, there is a dedicated team which does the user acceptance testing. They prepare their own test cases, perform them and validate them. On successful completion of the UAT, the developer is notified of the results and sign off is given to go ahead with the installation in production. In case of defects during UAT, they are logged in the tool, specified by the client shop and assigned to the developer for being fixed. The developer performs the check out process, fixes the code and checks in back the component for the team to continue the UAT testing.

14

Development on System z

According to Gartner, there are 200 billions lines of COBOL code that run today's businesses worldwide. Fortune 500 companies maintain 500 million lines of code and add 10% more new line of codes annually to the existing mainframe applications to keep up with business needs, changing conditions, and regulations.

For heavy-duty transaction processing, nothing can beat mainframe, and thus most of the fortune 500 companies in financial services, retail, and airline rely on mainframe for their millions of concurrent transactions. Because most of their nucleus application runs on mainframe, often there is a call to rewrite the whole application or addition of new modules or a significant maintenance activity to add new programs and interfaces. In all these cases, there is a scope of development.

In the twenty-first century, though it is alluring and inherent to think of 4GL language when it comes for development, it is equally difficult to take a decision for any stable application that has run in mainframe for decades with no major disruption. And thus still today, there are few new developments executed on mainframe involving green screen by many industries and only a few companies still develop tools for the optimization of the development life cycle. In the Java/J2EE and Web 2.0 world, there is a huge rush to give a face-lift to the existing legacy applications in either integrating with distributed applications or modernizing the application in terms of technology, interface, and so on.

14.1 What is mainframe development

In this section we discuss various considerations about mainframe development.

14.1.1 Introduction

Development in the IT world means the activities required to realize a business requirement into a software application. The activities must touch all the phases of the Software Development Life Cycle (SDLC). Development starts with a high-level business requirement document and ends with the code and user guide delivery. Due to the high cost of the system, only sizeable companies opt for mainframe and develop core business applications that require heavy duty transactions and complex workflow. Thus mainframe application development is much complex in nature. Both client and the IT service provider play vital role in a development success story. The key elements those need to be given focus are:

From the client side:

- Choosing the right architecture and framework that can be scalable
- Selecting the right transaction server, database, and programming language combination that are optimized to address the business need

From the development team:

- Matured requirement capture process and a stringent change control board
- Vigilant project management
- Expert technical team members on the chosen technologies
- Right IDE to debug and performance testing of the application
- Stress on functional and load testing

> **Tip:** Always in a development project, requirement management is a critical issue. It is a new baby and clients want to have everything at once, irrespective of the project phase. The project manager and change control board have to be mature enough to make proper decisions and suggest to the client the iterative delivery method for better management and timely delivery.

14.1.2 Enhancement versus development

The life cycle of the execution is same for enhancement or development of a software application. The major difference is, development is from green-field and enhancement is based on a blue print or an existing application. When there is a continuous or significant change in the existing application, it calls for an enhancement.

The types of enhancements are as follows:

1. Additional Module development:

   It is for small change requirement(s). It involves additional batch program or interface.

2. Application rewrite:

   – Mostly it is required when there is a change in line of business or a new acquisition.

   – It involves writing the application from scratch; reference might or might not be taken from the existing application.

3. Application modernization:

   – It is required to integrate the core legacy applications with the distributed enterprise world.

   – It can be done in three ways:

     • Hooking up the legacy database or file structure through JDBC or Web Service

     • Accessing the legacy applications through Web Service

     • To and fro asynchronous data flow between legacy and open system through middleware

> **Tip:** With the evolution of System z and continuous focus in integrating the 4GL languages and enterprise technologies, it is extremely difficult for a mainframe shop to take the right decision on a development call. Before taking any decision for change, organizations have to focus on the proven technology and methodology, own strength, best ROI, and most importantly, the contingency planning.

14.1.3 Industry trends in mainframe development

Unless the organization deals only in mainframe or takes a very conservative approach in changing the existing run-in-the-mill applications, there is a paradigm shift in the development approach. There are few "cent-per-cent" mainframe green-field developments today. There are different tiers (data, application, and presentation) of components involved in the development or enhancement life cycle.

For any new development in the mainframe shops, consider these possibilities:

Data Tier: Still remains in the mainframe system, preferably in terms of:

- Stored Procedures those can be hooked by application run in any platform
- Data exchange in terms of XML

Application Tier: Application development, gives emphasis on:

- Component based design and development
- Usage of both structured and OO languages and compiled under LE
- Usage of integrated tool for to create platform independent services and to build traditional and composite applications, including CICS, IMS, COBOL, PL/I, Java, EGL, Web, Web 2.0, and XML-based applications.

Presentation Tier: Designed keeping in mind:

- The layout to be used by different devices
- Examples include the CICS screen used by dumb terminals or Java based interface run on a mobile browser

14.1.4 Business decision: Development/integration or modernization

In this section we discuss the following scenarios.

Case 1

This case involves the following considerations.

Business requirement

The core system of an Insurance company PrimeInsure runs on a mainframe comprising CICS, COBOL, VSAM, and DB2. It acquires the company MyInsure. The line of business of MyInsure is different and a few features are really very attractive. The core business process is almost same in both the cases except for a couple of modules. PrimeInsure has a lot of senior mainframe programmers who can develop any new functional screen and application quickly.

Development decision

Keep these considerations in mind:

1. The applications of MyInsure Company must be included under the core business category. So development of an external interface is ruled out.
2. The business application of MyInsure Company needs speedy data entry and retrieval. Web interface will be a challenge in terms of response time.
3. PrimeInsure Company has seasoned mainframe programmers and they have been in the system for decades. To develop any 4GL interface, the company needs to acquire developers from market and provide the time to get familiarized on the business and application.

Solution

Design of the application on Web Technology or using 4GL is ruled out. It is a Development case where the existing infrastructure, applications, and resource can be leveraged. The experienced mainframe programmers are aware of the intricacies of existing business. Given a little time to analyze the application of MyInsure Company, they can come up with the requirement for what can be re-used and what to develop additionally. So PrimeInsure Company takes the decision to develop the additional interfaces involving the existing programmers on mainframe technologies such as CICS, COBOL, VSAM, and DB2.

Case 2

This case involves the following considerations.

Business requirement

MotorUS manufacturing unit has a full-fledged mainframe application to take and execute the order. The company decides to provide a new web front-end for order maintenance for its internet and mobile users.

Development decision

Here are a few considerations:

1. For internet and Mobile users, there is no choice other than development of a web interface for the presentation tier.
2. To simplify the integration, and to minimize the effort and roll-out time, architecture must be chosen to reuse the existing components.

Solution

This is a Modernization case, which can have many approaches to implement.

Approach 1:

- Through the Host Transformation Service, the existing mainframe 3270 online screen can be transformed to an intuitive Web, portlet, rich client or mobile device user interfaces (Figure 14-1).

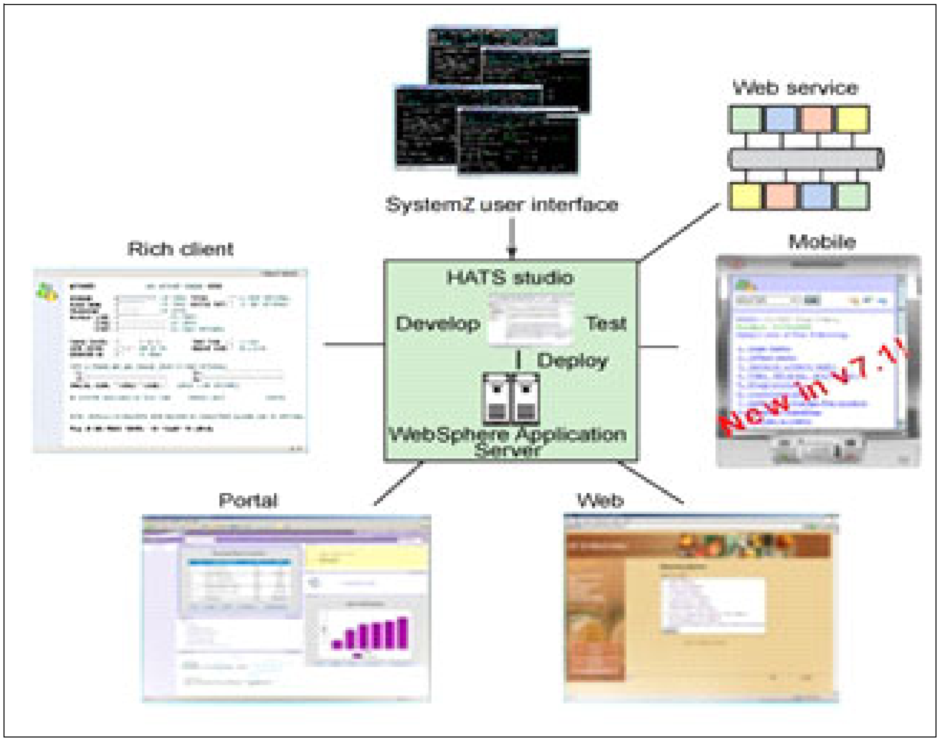


Figure 14-1 Host Transformation Service

Approach 2:

1. Develop a web interface for presentation tier
2. Leverage the existing application and data tier. Through a Web Service and CICS Transaction Gateway, connect the CICS transaction from the Web interface (Figure 14-2).

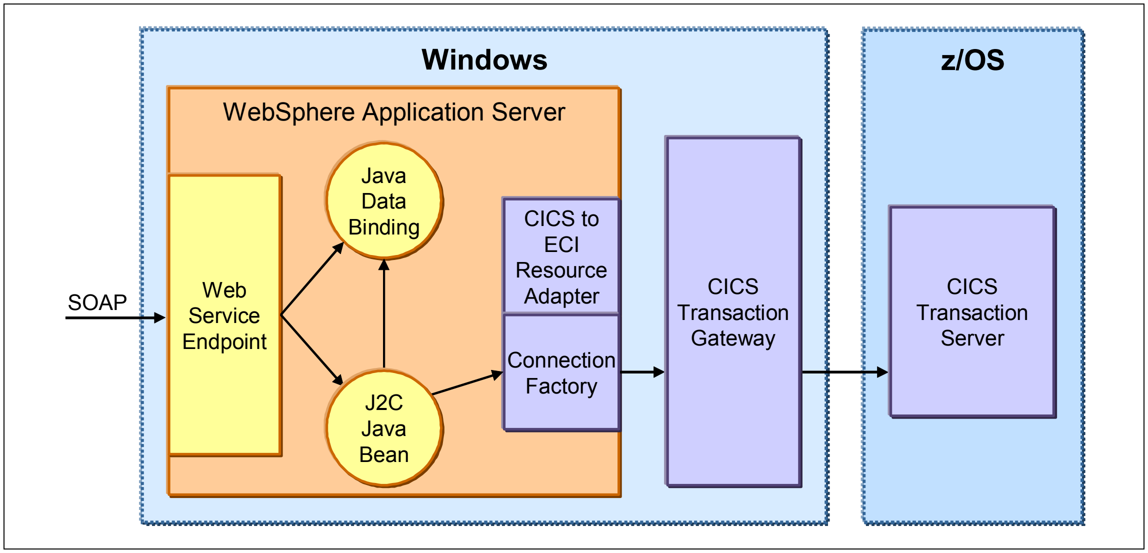


Figure 14-2 CICS Transaction Gateway

Case 3

This case involves the following considerations.

Business requirement

Major applications for ABC Company run on S/390 mainframe. Only one application runs on AIX for trading partners. There is a requirement to synchronize the trading life cycle through these two systems.

Development decision

Here are a few considerations:

1. Need to decide whether the synchronization needs on Synchronous / Asynchronous manner
2. If needed in a synchronous way, then the Web Service can integrate both the application as mentioned in the Case-2.
3. If it can be manageable in an asynchronous way, then communication through MQ is the best solution.

Solution

This is an integration case. In the asynchronous mode of communication between two of the systems, it can be performed by MQ. One MQ sits on the AIX server and the other on the OS/390 side. Both are connected through MQ channel. An AIX box MQ can push data to mainframe MQ and vice versa (Figure 14-3).

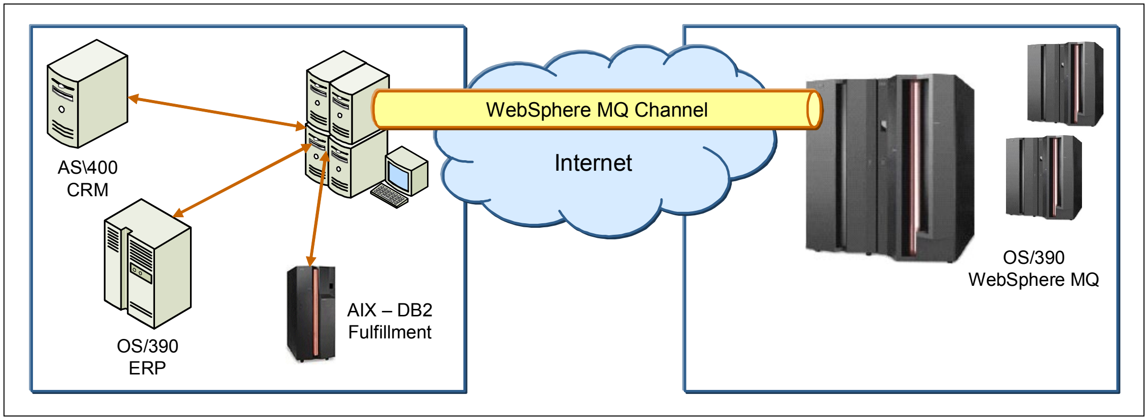


Figure 14-3 Integration through MQ

14.2 Key attributes of mainframe application development

14.2.1 Focusing on harvesting the assets with emphasis on re-use

Before starting any application development, the best practice is to have a deep dive into the existing application and to find out if any component can be re-used. In addition to reducing the time spent on development effort, it also avoids building another similar component in the application portfolio. This approach facilitates leveraging any existing behaviors (rules, transaction flow, logic, and security) of the application without having to re-invent it.

14.2.2 Greater application heterogeneity: COBOL, Java, middleware [MQ], WSDL, SOAP

Any existing enterprise applications for bigger corporate houses are interconnected like spaghetti. Any development work needs a lot of due diligence and brainstorming in terms of platform, architecture, and language selection. Selection must be based on the application's characteristics of scalability, reusability, and interoperability.

14.2.3 Faster time to production through iterative development

Unlike the past, now the time-to-market is expected in weeks rather than in months or years. So for any development, the focus must be on lesser turn-around time by choosing an optimized process or methodology such as Agile for the development approach.

14.2.4 Squeezing development time through right IDE

To speed up the development, there must be a right IDE to take care of all COBOL development tasks (design, coding, testing, and debugging) in the open and extensible framework. The IDE must have:

- A scalable Eclipse-based environment enabling development and deployment on a range of platforms
- A real-time syntax checker to validate the errors during coding and not to wait till compilation
- An integrated debugging environment
- A template based framework for different technology combinations

14.3 System z application development approach

In this section we consider the following development approaches.

14.3.1 Traditional approach: The waterfall method

Traditionally, and even today, methods and processes used in the System z development environment are generally based on a waterfall life cycle model.

Example of the waterfall approach

For example, we generally adopt a workflow that progresses sequentially from Requirements gathering, on to Analysis, and then through Design, Code and Unit Testing, through to Testing and Deployment (see Figure 14-4 and Table 14-1).

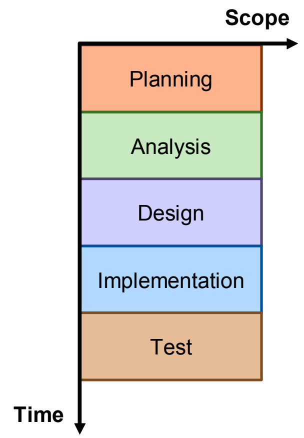


Figure 14-4 Software development through waterfall method

Table 14-1 Activities at different phase of a development requirement

Phases	Activities
Project Planning	► Time spent prior to gathering requirements on activities such as defining the scope of a project, establishing the team and project champions, etc. ► Time spent on planning and tracking activities throughout the project's life. ► Training specifically required for the project. ► Evaluating and selecting tools, computers, operating systems, and so on for a specific project
Requirement	► Working with customers to define the functions that must be contained in the system. ► Writing and inspecting the software requirements specification. ► Doing any necessary rework as a result of the inspections. ► Creating and evaluating prototypes intended to more completely specify the requirements of the system.

Phases	Activities
Design	► Creating and inspecting high-level design models, such as data flow diagrams, entity relationship diagrams, and dialog maps. ► Doing any necessary rework as a result of the inspections. ► Designing user interface screens, algorithms, data structures, file formats, database schemas, and classes. ► Creating and evaluating prototypes intended to determine whether the proposed system design is correct. ► Defining program architectures. ► Program design, including writing minispecs or PDL. ► Evaluating, by whatever means, alternative solutions to any technical problem or user requirement.
Implementation	► Coding a program. ► Inspecting the source code. ► Doing any necessary rework as a result of the inspections. ► Writing unit-level, internal module documentation. ► Creating database tables, user interface screens, data files, or other artifacts that are part of the final system. ► Correcting defects found during testing, but prior to delivery of the system.
Testing	► Writing and executing unit tests in a disciplined and systematic way. ► Informal, ad hoc testing of the program. ► Writing and inspecting software quality assurance plans and test plans. ► Doing any necessary rework as a result of the inspections. ► Writing and executing user interface or other system tests. ► Designing and implementing automated test drivers and associated test data files. ► Engaging in systematic alpha or beta testing. ► Recording the results of formal test executions.
Writing Documentation	► Writing user aids (on-line help, user guides, reference manuals, tutorials, training materials). ► Writing system documentation (i.e., external to the programs themselves). ► Writing internal reports or giving presentations on the project.

Deficiencies of the waterfall approach

The following considerations apply:

1. The requirements discipline in the waterfall model is usually performed one time at the beginning of the project based on which product or application is built. A considerable amount of time elapses while deriving the requirements, building product, and getting it tested and delivered. By then, more likely, user needs have changed in order to keep pace with the current climate of frequently changing business conditions. So the primary problem here is the inability to easily adapt to changing user requirements.
2. In waterfall approach, the deliverable is handed over to the user in the form of the final product, which is delivered at the very end of the development cycle. This practice, as alluded to earlier, creates cause for surprises because user needs are not accurately met.

There are multiple methods that use the iterative approach. The two most frequently used methods are the Rational Unified Process (RUP) and Scrum.

14.3.2 Collaborated approach: The iteration method involving RUP

The Rational Unified Process (RUP) for System z addresses “green-field” development, as well as system evolution that involves architectural changes (including turning an existing capability into a Web Service, for instance) or significant impact on existing business processes. However, pure maintenance activities are outside the scope of RUP for System z.

For more information about using RUP on maintenance projects, refer to the IBM Rational Unified Process for Maintenance Projects, which is a Rational Method Composer 7.1 plug-in that describes a delivery process for performing maintenance on a software product. The plug-in provides a delivery process, tasks, and guidance for understanding the purpose of a maintenance development cycle, avoiding pitfalls during a maintenance cycle, and successfully delivering a product with higher quality than the previous release.

RUP gives the iterative development approach to a new dimension. The key developments are as follows:

- Requirement is done not only at the beginning but continues throughout the process, because requirements by nature change over time and development efforts need to be aligned to stakeholder needs.
- Implementation starts earlier to enable early stakeholder feedback, which is key to ensuring that we build the right system.
- Test starts earlier, because the later the discovery of defects, the more costlier it is to put things right.

- Project plans are refined throughout the project based on a continuous re-evaluation (at least one time per iteration) of risks and priorities.

Development RUP phases

The RUP phases are illustrated in Figure 14-5 and described in Table 14-2.

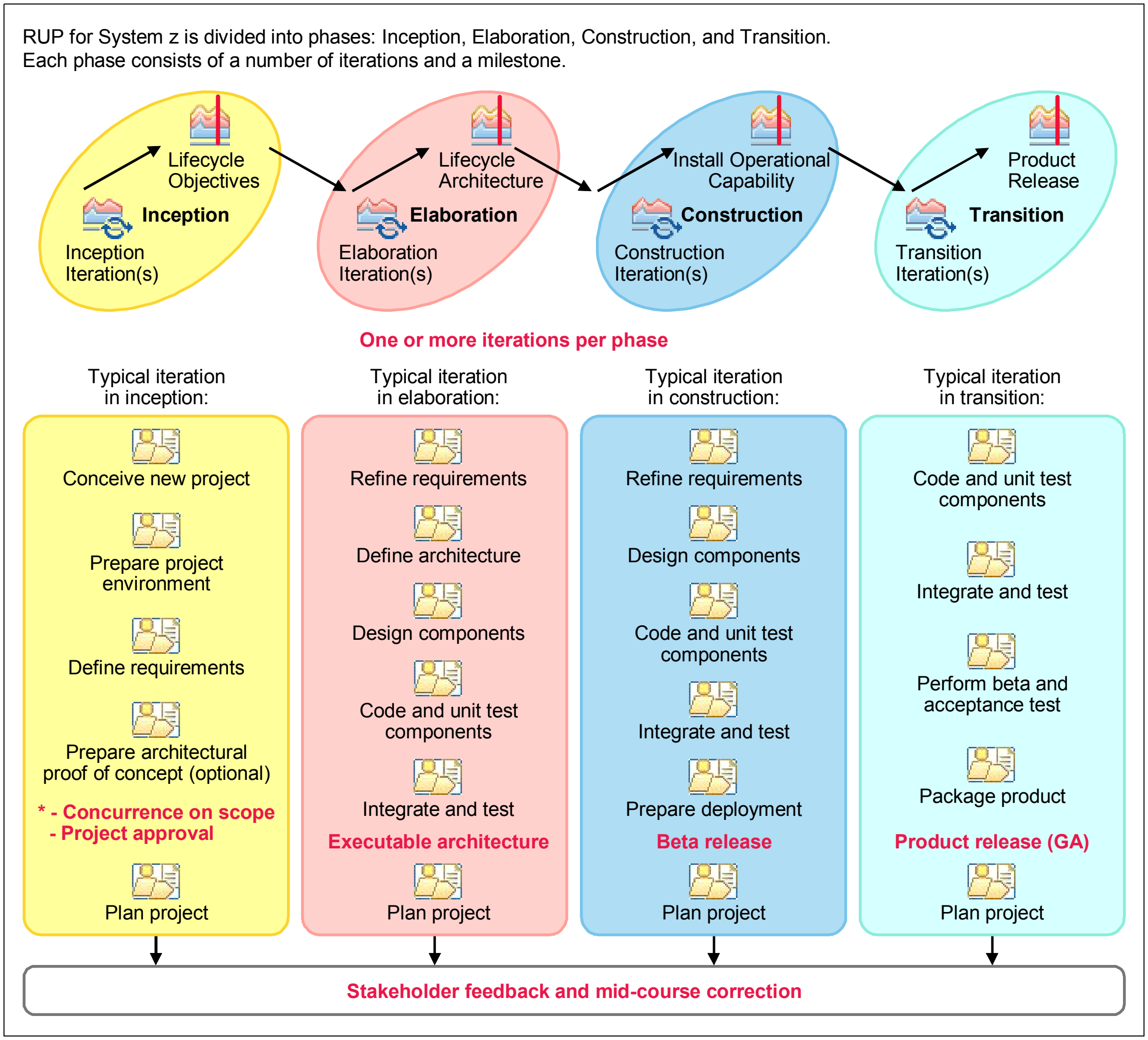


Figure 14-5 Software Development through RUP Method

Table 14-2 RUP activities and tools

Phase	Activities	Rational Tools used
Inception	► Stakeholder concurrence on the scope definition. ► Right set of requirements has been captured. ► The cost/schedule estimates, priorities, risks, and development process are appropriate. ► All risks have been identified and a mitigation strategy exists for each risk.	► Rational Software Architect/ Modeler - To create the models ► IBM Rational RequisitePro® - To manage requirements ► Rational Portfolio Manager - To create the software development plan and detail iteration plans ► Rational Clear Case - Used for Configuration management
Elaboration	► Requirements are captured and signed ► Architecture is defined ► Model, design, interface, and prototypes are done and tested ► Unit and Integration test is done for the prototype	► Rational Software Architect/ Modeler , Rational RequisitePro, Rational Portfolio Manager, Rational Clear Case ► Rational Developer for System z - To develop and do the unit of the prototype
Construction	► Complete the development based on the baseline architecture and prototype ► Coding and Unit test is done for all the components ► Installation and Deployment plan is ready	► Rational Software Architect/ Modeler , Rational RequisitePro, Rational Portfolio Manager, Rational Clear Case, Rational Developer for System z ► Rational Manual Tester - To exercise the test cases ► Rational Functional Tester - To automate the testing suite
Transition	► Testing the product for release ► Making finer adjustment based on the user feedback	► Rational Developer for System z, Rational Manual Tester, Rational Functional Tester, Rational Portfolio Manager and Rational Clear Case

RUP: The deliverable dashboard

The dashboard is shown in Figure 14-6.

Work Product	Inception	Elaboration	Construction	Transition
Business Case	100%			
Vision	100%			
Glossary	40%			
Software Development Plan	80%	95%	100%	
Iteration Plans	E1 100%			
Risk List	25%	50%	75%	100%
Use Case Model	20%	80%	100%	
Supplementary Specification	20%	80%		
Software Architecture Document	10%	100%		
Analysis Model		50%	100%	
Design Model		60%	95%	
Service Model		60%	95%	
Test Plan	10%	30%	90%	
Test Cases & Test Scripts		40%	80%	100%
Source Code			95%	
Installation Verification Procedures		80%		

Figure 14-6 Deliverable Dashboard

14.3.3 Collaborated approach: Iteration method using Scrum

Another important form of iterative development is Scrum. In this method, an iteration is performed for each new feature added to the system, and at the end of each iteration, there is a working form of the product.

Scrum method phases

There are four phases in the Scrum method: planning, architecture, development sprints, and closure. In the planning phase, the developers define the changes to be made, plan the schedule, and estimate costs. In the architecture phase, they plan how the changes will be implemented. The development sprints consist of developing the new functionality. Finally, during the closure phase, the developers plan for release. See Table 14-3.

Table 14-3 Scrum roles

Roles	Planning / meeting	Artifacts
► Product owner ► Scrum master ► Team member	► Daily Scrum ► Sprint Planning ► Sprint Review ► Sprint Retrospective	► Product Backlog ► Sprint Backlog ► Burndown Chart ► Impediments List

Scrum framework workflow

Requirements in a Scrum project are captured as items in the Product Backlog

- Items from the Product Backlog selected for completion in the sprint planning becomes sprint backlog.
- A sprint backlog contains specific tasks to be performed.
- During the sprint, the status of the task is updated daily, which keeps Sprint burn-down chart updated.
- At the end of each sprint, the accomplished work is shown in the Sprint review meeting.
- After the sprint review, the meeting Sprint retrospective is planned to assess the status and to revisit the approach of the plan for the next sprint.
- The Scrum master maintains the Impediment list, if any, for the Scrum development.

When managing a project by using the Scrum method, you need a place to keep the description of the tasks that need to be done for the project, namely, your stories and tasks. In a Project Area, you can manage your builds, sources, plans, and much more. This section focuses on the planning and management aspects of the Scrum process.

14.4 Next generation development using Agile Methods

In this section we discuss the use of Agile Methods.

14.4.1 What are Agile Methods

Industry and technology move extremely fast, requirements change at rates that swamp traditional methods, and customers have become increasingly unable to state their needs up front while, at the same time, expecting more from their software. These changes in the software industry led to the development of Agile Methods, a form of iterative development. Agile Methods are a reaction to traditional ways of developing software and acknowledge the need for an alternative to documentation driven, heavyweight software development processes (Figure 14-7).

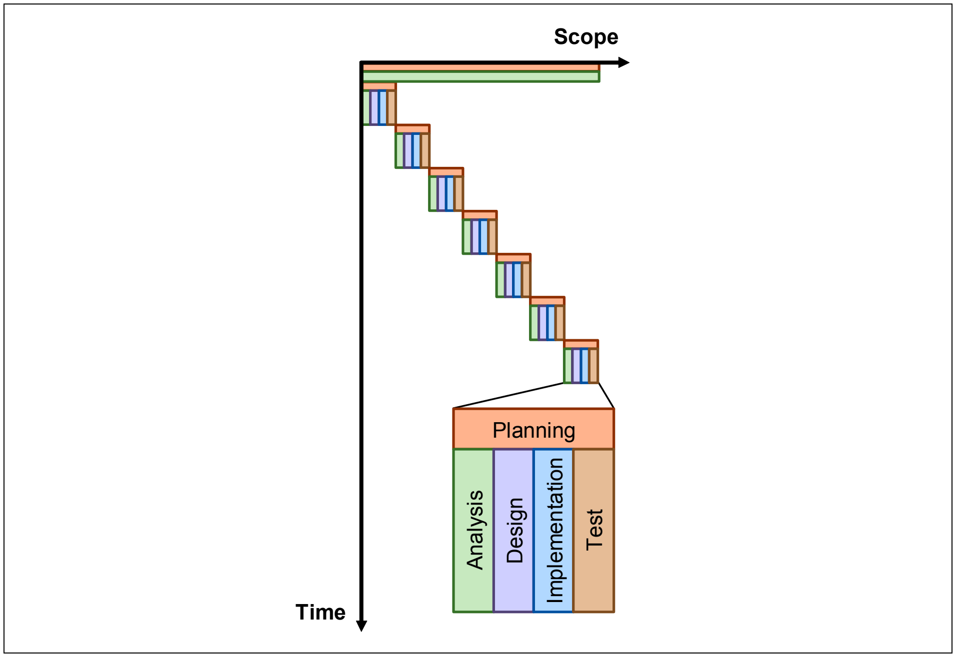


Figure 14-7 Software Development through Agile method

Agile development is a group of iterative software development methodologies that includes Extreme Programming or XP for short, Scrum, Standard & Poor's, Feature Driven Development, Crystal, and Adaptive Software Development.

All Agile development methodologies share the following features:

- Individuals and interactions are valued over processes and tools.
- Working software is valued over comprehensive documentation.
- Customer collaboration is valued over contract negotiation.
- Responding to change is valued over following a plan.

14.4.2 The IBM solution through RTCz

The integration of an Agile planning tool with IBM Rational Team Concert and IBM Jazz™ provides greater visibility, opportunities for collaboration, and traceability as well as process awareness into a single platform for improving development productivity.

RTCz developed on JAZZ platform supports the Scrum framework for the mainframe development on agile methodology. The Scrum process helps ensure that the most valuable remaining features are built next, and it emphasizes that work will always be completed to the point of being deliverable. Work is structured in two-week to four-week sprints, or iterations, with working software produced by the end of each sprint.

RTCz provides a collaborative development as follows:

- Configuring with the existing configuration management system
- Using IBM Rational Build Forge® to perform the agile development for host-based project
- Using the rational integrated tool RDz (Rational Developer for System z)
- Setting up components and streams for optimized cross-system development

Rational Team Concert for System z is based on a team server component that provides the following advantages:

- Collaboration and process capabilities
- A repository to store the data, including any related assets
- A rich Eclipse-based client for developers
- A Web-based client for project status-checking and occasional users

Mainframe development is done using Scrum methodology on RTCz:

- Creation of a SPRINT and Project Area
- Assign of Team Member to tasks of a SPRINT backlog
- Team Member Dashboard - Status vs. Task Distribution

A

IBM Rational Developer for System z productivity benchmark comparison

This appendix documents a benchmark that was completed in March 2010 that compared two IBM products: IBM Rational Developer for System z V7.6.1 and Interactive System Productivity Facility (ISPF) V6.0. The study consisted of a set of tasks normally associated with traditional z/OS COBOL maintenance and production support:

- Analyzing the paragraph flow and program structure of code
- Navigating and modifying existing COBOL programs
- Adding a small amount of new business logic to an existing program
- Updating fields in copybooks and modifying code that referenced these fields
- Performing traditional data flow analysis
- Working with SQL and DB2 test data
- Compiling and linking a program

The detailed scripts used by participants in the study are on the ITSO server:

`ftp://www.redbooks.ibm.com/redbooks/SG247868`

Reasons for the study

Back in the early 1990s, there was a noticeable shift where writers argued that companies needed to downsize their production workloads off mainframes to distributed platforms. Unrealistic claims were made, which were not analyzed, and entire websites now exist that document the cost and other unintended consequences of wishful thinking posing as research.

The idea that you can replace a mature, stable, scalable, powerful, manageable centralized hardware platform with new, less stable, and harder to manage and support server farms and complex application stacks does not make sense. This idea simply does not account for the complexity, scale, and scope of production workload file and database I/O sizing and transactions and batch window throughput.

Except for extremely small shops or for small departmental applications, the most cost-effective place to run mission-critical and performance-sensitive enterprise applications is on a centralized System z server.

However, developing mainframe software off the mainframe has been an attractive concept for several decades. Development seems both practical and productive, given today's desktop technology. And, of course, mainframe software development and maintenance are both practical and possible with the tools of the optimized life cycle, but is it really more productive? What tangible evidence exists that proves Rational Developer is actually a more productive development platform than green screen (ISPF)? And if so, exactly how much more productive? And, for whom is it more productive?

IBM management asked us these questions in late 2009. Specifically, they asked us to design a comparison study between two integrated development environments (IDEs):

- ISPF Version 6.0 running on an IBM System z 10
- Rational Developer V7.6.1 running on a windows thinkpad

We conducted the study using this criteria:

- Comparable IDEs
- Sizeable number of participants
- Focus on common, everyday z/OS traditional maintenance tasks
- Scientific research project with double-blind controls and so forth

Study design

In attempting to satisfy the preceding criteria, the IBM team started by polling a number of ISPF/COBOL programmers to determine what kinds of activities were consistent on a daily basis. The ISPF/COBOL programmers were IBM employees, IBM Business Partners, and clients.

We created a large task list of 100 discrete activities for ISPF and verified them with the ISPF/COBOL programmer team. We divided the 100 tasks into seven categories:

- Code navigation
- Edit operations (basic ISPF edit operations)
- COBOL coding: Adding a new paragraph to an existing COBOL program with changes to the copybooks
- Data flow analysis: Following the value in a variable as it is moved
- Code and syntax error removal
- Build: Compile and link
- DB2 and SQL work: Modifying DB2 table values, creating new rows, writing SQL statements, and testing SQL statements

Obviously, this list is not an all-inclusive list of z/OS developer activities. We did not include Virtual Storage Access Method (VSAM) and data file maintenance and debugging. We vetted this list with the ISPF/COBOL programmer group. However, we encourage you to download copies of the scripts to see for yourself:

ftp://www.redbooks.ibm.com/redbooks/SG247868

Avoiding human factors

Every attempt was made to remove human factors and bias from the research by using the following approaches:

- We followed closed end, click-for-click detailed instructions (Figure A-1 on page 591 and Figure A-2 on page 592) to minimize:
  - Think time
  - Differences in product (ISPF or Rational Developer) experience, typing speed, and application development experience
- Project participants were told that they were trying to find gaps between Rational Developer and ISPF functionality, which was a single-blind research tactic that attempted to mitigate subliminal bias

- Half of those participants that followed both the Rational Developer and ISPF scripts followed the Rational Developer script first, and the other half followed the ISPF script first. We reversed the order with the second half of the participants to mitigate “learning and retention” bias.

ISPF bias

Two areas of the study were biased toward ISPF and against Rational Developer:

- The scripts were written from an ISPF perspective, not a Rational Developer perspective. We created the scripts based on the daily actions of ISPF developers, and we adjusted the Rational Developer script to functionally match the actions that were done with ISPF. This approach biased the results against what might have been achieved if we had started from a Rational Developer perspective and tried to match ISPF to the product’s capabilities.
- We optimized the way that the tests were administered for ISPF, not Rational Developer. Because the scripts were exceedingly long (we documented the 100 steps for ISPF in 676 rows in a spreadsheet, at 10 point font), we decided that printing hardcopy and reading did not work. So instead, we told the participants to display the script alongside ISPF or Rational Developer and to read and scroll. This approach worked well for ISPF, because most participants were in 3270 mode, and ISPF fit perfectly in full screen view. However, they had to minimize Rational Developer, which required extensive horizontal scrolling. This approach minimized many of the benefits of the Rational Developer Eclipse Microsoft Windows orientation, because developers only viewed two-thirds of the product.

Participant characteristics

There were 23 participants in the study:

- IBM Business Partners
- Clients
- Academics
- IBM internal employees:
  - ISPF consultants, mostly IBM Global Business Services® professionals
  - Rational technical field personnel

The participants had the following average number of years of experience:

- ISPF: 12.7 years
- Rational Developer: 1.3 years

Summary

Using these control mechanisms, we made every attempt to meet the research goals that IBM management set. For the most part, the standard deviation statistics showed that we met these goals. See Figure A-1 and Figure A-2.

However, do not interpret these benchmark results as academic-level research. We obtained all the performance data that is shown in this appendix in the specific operating environment and under the conditions described in this publication, and we present this data as an illustration only.

Performance that is obtained in other operating environments might vary and we advise that you conduct your own testing.

```
From within ISPF
Open the dataset: <HLQ>.BNCHMRK5.JCL
Again Edit the member: SANDBOXI
    - From the command line type the following:
          C SANDBOX  SANDBOXI  ALL
          SAVE
          SUB
         -- and note the JobName produced by TSO.
From the command line, type: =3.8
   Open the JES job name and Job number for the compile you just
   submitted (note that the JCL Job name is <HLQ...I>
   M/PF8 to the bottom of the file
   View the syntax return code
From the command line, type:  =1   - to bring up ISPF browse
Open the file:  <HLQ>.BNCHMRK5.LISTING(SANDBOXI)
From the command line, type:  M  and press PF8
This will bring up all of the syntax errors in succession
With your cursor on the command line, press PF2
Open <HLQ>.BNCHMRK5.COBOL(SANDBOXI) in ISPF edit (=2)
Fix all of the syntax errors in SANDBOXI - by cross-referencing the
compile listing with the SANDBOXI program.  Because we don't want you
to guess, here are the identifier name mis-spellings:
    VLAD-RECORD should be VALID-RECORD
    READ PATMST should be READ  PATMSTR
    GO TO 100-ABEND-RTN should be 1000-ABEND-RTN
    INPATIENT-REC-ERROR should be INPATIENT-TREATMENT-REC-ERR
When you've fixed all of the errors (or think you have, and are ready
to re-compile), from the command line, type SAVE <Enter>
Press PF9 to swap to the split screen
```

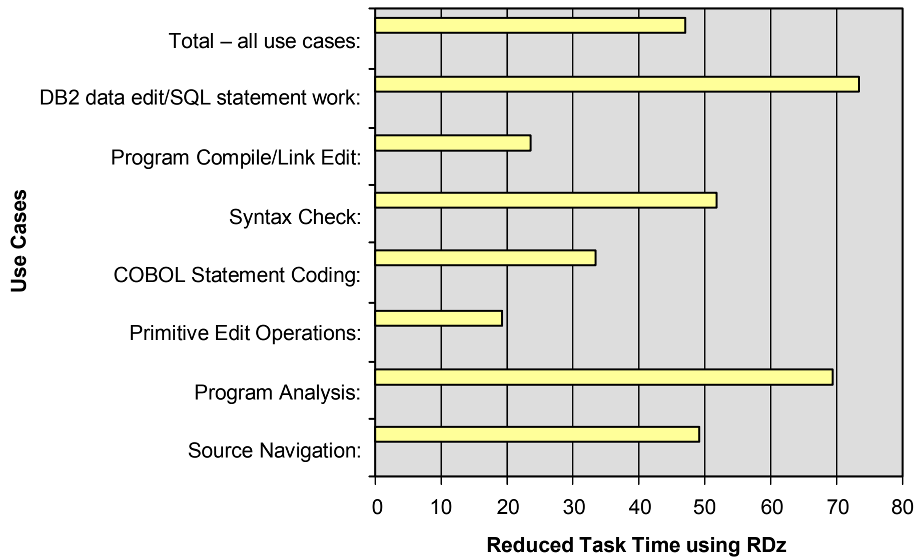
Figure A-1 Detailed script for ISPF tasks

For this step and the next, you will need to create an MVS SubProject, and populate it with the **BNCHMRK6.COBOL** and **COPYLIB** PDS's - see screen capture ==>
From within your MVS Subproject, open the file: **SANDBOXR**
Open the dataset: **<HLQ>.BNCHMRK6.JCL**
Edit the member: **SANDBOXR**
Right-click and select: **Syntax Check** > **Remote**
A job will kick-off on the host. When it finishes, open the Remote Error view to see the Syntax Errors produced. There are a large # of warnings about DB2 pre-processing and directives at the top of the view. Ignore these - and head straight to the Red-X syntax problem lines.
Fix all of the COBOL syntax errors in SANDBOXR - by double-clicking on the line in the Remote Error view, and reading the text supplied. Because we don't want you to guess, here are the identifier name mis-spellings: VLAD-RECORD should be **VALID-RECORD** READ PATMST should be READ **PATMSTR** GO TO 100-ABEND-RTN should be **1000**-ABEND-RTN INPATIENT-REC-ERROR should be **INPATIENT-TREATMENT-REC-ERR**
When you've fixed all of the syntax errors, 1. Right-click and remove all of the Error Messages from the Remote Errors View 2. Save and Syntax Check > Remote

Figure A-2 Detailed script for Rational Developer tasks

Task summary results: All participants subset

We entered the data from individual timings (down to the second) for each participant performing each task into a spreadsheet, and we graphically summarized the results (Figure A-3) as the percentage less time it took for all participants to complete the 100 tasks. In the bar chart, zero (0) represents the ISPF baseline default.



Use Case	% Less time to complete tasks with RDz				
Source Navigation:	49.26				
Program Analysis:	69.67				
Primitive Edit Operations:	19.22				
COBOL Statement Coding:	33.11				
Syntax Check:	51.89				
Program Compile/Link Edit:	23.38				
DB2 data edit/SQL statement work:	73.41				
Total – all use cases:	**46.88**				

Figure A-3 Results for all participants

You might note that certain categories of tasks showed more or less productivity, but overall, the results were favorable for Rational Developer across the board.

We analyze these results later, but before doing that, note that we also broke out the participants into a subset of what we called Time Sharing Option (TSO) Top Gun developers.

The TSO Top Gun developers characterized themselves with these attributes:

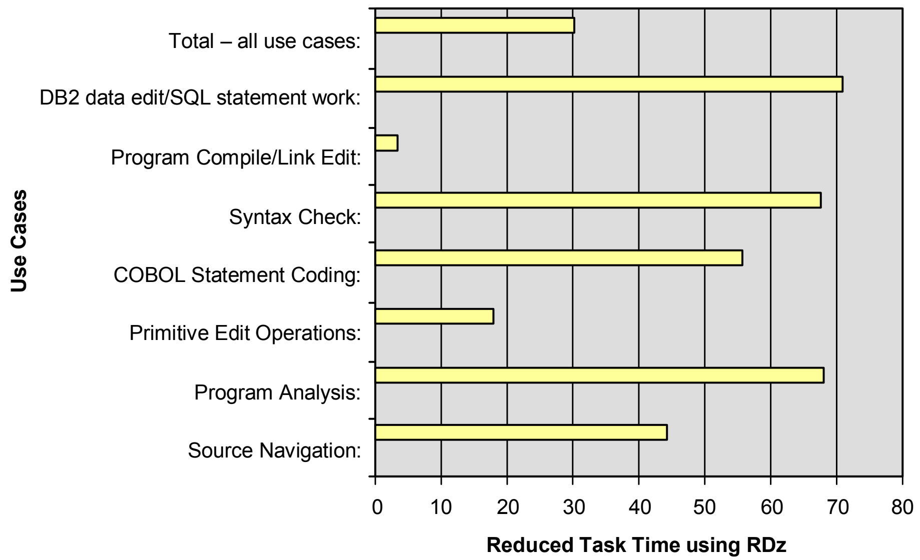
- Current with ISPF
- At least 20 years of ISPF experience
- Accurate and fast typing skills

We felt that these individuals represented a specific highly experienced ISPF-oriented constituent subset of most z/OS shops, so we created another set of summary statistics for them (Figure A-4).

Task summary: TSO Top Gun participants

From Figure A-4, you can see that the TSO Top Gun developers lived up to their nickname. Their results were better (and often significantly better) in most work categories than the pool of all developers (which represents in any given shop your entry-level programmers combined with TSO Top Gun-type users).

Most shops have a mix of experienced and entry-level programmers. The entry-level programmer component will increase more and more as the experienced programmers retire. The standard deviation for the Top Gun group was, across the board, in the single digits and often as low as 2 or 3 (showing consistent and statistically significant results).



Use Case	% Less time to complete tasks with RDz
Source Navigation:	44.53
Program Analysis:	67.99
Primitive Edit Operations:	17.45
COBOL Statement Coding:	55.98
Syntax Check:	67.58
Program Compile/Link Edit:	3.37
DB2 data edit/SQL statement work:	70.92
Total – all use cases:	**30.03**

Figure A-4 TSO Top Gun results

Analysis and feedback from participants

After the participants completed their work and returned the spreadsheet, we asked both close-ended and open-ended questions regarding what they thought the reasons were for the Rational Developer productivity.

The results were a little surprising but uniform. There were four sources of productivity:

- Significantly less typing with Rational Developer
- Integrated, language-sensitive, hyperlinked, feature-rich tooling
- Better use of the screen area through Windows
- Responsive desktop/windows environment

It is usually assumed that the superior engineering that goes into new technology is the primary reason for improvements. At least in the case of this study, the engineering was definitely a factor, but there were also many other positive factors or unintended consequences of Eclipse and Rational Developer development.

Significantly less typing with Rational Developer

Every activity on ISPF requires a degree of typing, which is typically custom typing (unique Find/Change commands, line location, and so forth). Even navigation is done with typing (=3.14, =P.DB2.3, and so forth).

Rational Developer has encapsulated most of those same actions and developer activities into declarative tooling (context menus, Intellisense and language-sensitive editing, and so forth), and the difference in the amount of required typing is significant. All twenty-three participants did both the ISPF and Rational Developer scripts, and they consistently listed “less typing” as their number one reason for increased productivity.

For example, ISPF development activities, which involve working with program copybooks and includes, require significantly more effort than Rational Developer, which offers context menu options to open and browse copybooks.

Integrated, language-sensitive, hyperlinked, feature-rich tooling

The green screen (ISPF) development environment is a manual and linear development process model, and the tools require constant panel navigation in order to access needed functionality. ISPF requires more typing and more MIPs with every PF key and Enter key pressed.

In ISPF, there is no hyperlinking of temporary results (for example, search and syntax errors) and no integration among or between the ISPF tools. To edit DB2 tables, you must enter long, cumbersome SQL INSERT or UPDATE statements.

Rational Developer integrates almost all common development functionality into a single system “concurrent development” view, and you can use the features and facilities without navigation (using context menus). Rational Developer hyperlinks source results whenever possible to further reduce extra navigation.

Better use of the screen area through Windows

ISPF allows developers to see from 24 to 55 lines of source at a time; however, there is a lot of wasted screen area and the quality of the source view is a problem when you work with 40 or more lines.

Rational Developer provides up to 190 lines of source viewing (and editing) in split-screen mode, with virtually no wasted screen space. Rational Developer allows vertical screen splitting, which is a more efficient way to handle 80-column COBOL statements.

With Rational Developer, you can use these functions:

- Copy and paste between open views of the same or separate source files.
- Edit the same program in two separate areas simultaneously, which is useful and impossible with ISPF.
- See the results of a change reflected across all open windows immediately.

Responsive desktop/windows environment

The IBM mainframe provided subsecond response time for the tests (even the compile and link jobs finished in less than a second for all participants). However, the ability to use the desktop environment (real-time scroll bars, Page Up and Page Down, and so forth) was still appreciably, if not significantly, faster for certain standard programming tasks, such as aligning code on the screen to specific statements, code navigation, and so forth.

This responsiveness was interesting, because it proved that downsizing away from mainframes was not ideal and that mainframes were definitely flexible and responsive enough. At least in this study, we found the mainframe power was a benefit, although it was the smallest factor in the Rational Developer benefits list.

Mitigating factors

Note the following mitigating factors about this benchmark.

We did not use custom ISPF Edit macros. Many shops (and individual programmers within shops) have developed and use custom editing macros during their work. These custom macros might improve the ISPF benchmark results, possibly as much as 5-10 percent. We did not use any ISPF Edit macros, because every shop's and programmer's macros are unique. However, everyone used ISPF option 3.4, option 2, option 1, split screen, TSO job submission, and so forth.

We also not use custom Rational Developer macros. In the same way, mature Rational Developer shops have either recreated their macro facilities in Rational Developer, or they have created unique extensions to the editor. These custom macros might have improved the Rational Developer results as much as 3-5%. As a short experiment, if you return to the results for the TSO Top Gun Developers (Figure A-4 on page 595,) and factor in a 10% improvement in ISPF productivity through macros and then factor in a 3% improvement in Rational Developer productivity (a 7% overall improvement), you will probably still find the results, a net 23% improvement in productivity, significant.

Also, the years of ISPF experience were a mitigating factor. The ISPF development experience (12.7 years) of the participants is considerably more than the Rational Developer experience (1.3 years). However, many shops have groups of developers with 20 plus years of ISPF experience. You might wonder how much more productive you get after almost 13 years of using a product; however, we wanted to call your attention to this issue. We mitigated this factor as much as possible through the use of the detailed ISPF script (down to the PF key to press) and the Rational Developer script (down to the context menu to use).

Summary

In this appendix, we introduced you to a recent study comparing two IBM products. We described the participant demographics, detailed the methodology, presented the findings, and analyzed the results.

The detailed scripts used by participants in the study are on the ITSO server:

ftp://www.redbooks.ibm.com/redbooks/SG247868

B

Additional material

This book refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

`ftp://www.redbooks.ibm.com/redbooks/SG247868`

Alternatively, you can go to the IBM Redbooks Web site at:

`ibm.com/redbooks`

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG247868.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks publications

For information about ordering these publications, see "How to get IBM Redbooks publications" on page 603. Note that several of the documents referenced here might be available in softcopy only.

- *Getting Started with SCLM: A Practical Guide to SCLM and SCLM Advanced Edition*, SG24-7392
- *CICS Explorer*, SG24-7778
- *Faster Application Change and Reuse with WebSphere Studio Asset Analyzer*, REDP-4225

Other publications

These publications are also relevant as further information sources:

- *Rational Developer for System z Host Configuration Utility Guide*, C14-7282
- *Rational Developer for System z V7.6.1 Host Configuration Guide*, SC23-7658-04
- *IBM Rational Systems Developer, Version 7.0 Installation Guide,* GI11-8355
- *Fault Analyzer for z/OS V8R1 User's Guide and Reference*, SC19-1253-05
- *File Manager for z/OS and OS/390 V2R1 User's Guide and Reference,* SC27-1315-01

Online resources

These web sites are also relevant as further information sources:

- IBM 2010 IDE efficiency study:

  `http://www-949.ibm.com/software/rational/cafe/thread/4056`

- Rational Developer education:

  http://www-01.ibm.com/software/rational/education/

- Rational Developer course:

  https://www-304.ibm.com/jct03001c/services/learning/ites.wss/us/en?pageType=course_description&courseCode=RN100

- COBOL Cafe:

  http://www-949.ibm.com/software/rational/cafe/community/cobol

- IBM Rational developerWorks site:

  https://www.ibm.com/developerworks/rational/

- Debug Tool for z/OS:

  http://www-01.ibm.com/software/awdtools/debugtool/library/

- Application Performance Analyzer for z/OS:

  http://www-01.ibm.com/software/awdtools/apa/library/

- File Manager for z/OS:

  http://www-01.ibm.com/software/awdtools/filemanager/library/#v101

- Fault Analyzer for z/OS:

  http://www-01.ibm.com/software/awdtools/faultanalyzer/library/

- To download CICS Explorer:

  http://www.ibm.com/software/htp/cics/explorer

- *Setting up the CICS management client interface*:

  https://publib.boulder.ibm.com/infocenter/cicsts/v4r1/index.jsp?topic=/com.ibm.cics.ts.clientapi.doc/topics/clientapi_setup.html

- MQ Primer:

  http://www.redbooks.ibm.com/abstracts/redp0021.html?Open

- WebSphere MQ V7.0 features and enhancements:

  http://www.redbooks.ibm.com/abstracts/sg247583.html

- IBM WebSphere MQ library:

  http://www-01.ibm.com/software/integration/wmq/library/

- Download a trial version of Rational Asset Analyzer:

  http://www.ibm.com/developerworks/downloads/r/assetanalyzer/index.html

- *Rational Asset Analyzer Installation Guide:*

  http://publib.boulder.ibm.com/infocenter/rassan/v5r5/topic/com.ibm.raa.doc/PDF/admhins.pdf

How to get IBM Redbooks publications

You can search for, view, or download IBM Redbooks publications, Redpapers publications, web docs, draft publications, and additional materials, as well as order hardcopy IBM Redbooks publications, at this web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

z/OS Traditional Application Maintenance and Support

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

z/OS Traditional Application Maintenance and Support

Agile development

Cost containment

Productivity and efficiency

In this IBM Redbooks publication, we attempt to provide fresh insight into a problem domain that, in the authors' opinions, has been pushed to the back burner of technology writing for far too long—the domain of z/OS (traditional) mainframe maintenance and production support. Since the mid-1980's, outside of a few websites and publications, this still-critical area of software has barely even received lip service by the world of mainstream technology media. In a small way, we are attempting address this situation.

In this book, we provide information in "what and how to" sections on the value of z/OS maintenance and support—not the value of the software, which is hardly in question, but the value of the software developers, and how they collaborate, analyze, code, and test the applications, fixes, and enhancements under their responsibility. We present new 21st Century tools to help them achieve their goals more easily and effectively. These tools integrate and provide a 1 + 1 + 1 = 5 value-proposition, for companies that are still doing work the way they did when in the mid-1970's, when Gerald Ford was president of the United States.

We are also describing, to a lesser extent, how you can effectively integrate the new tools with your existing development software stack, in order to find points of complimentary functionality. And we describe the new agile development and maintenance methodologies, and best practices for tools use and adoption.

We hope that you find this work useful, and perhaps that it can fuel more discussion, future Redbooks publications, and other publications by IBM, or any vendor or group interested in this critical and vastly under-acknowledged technology domain.

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

SG24-7868-00 ISBN 0738435589