

Modernizing IBM i Applications from the Database up to the User Interface and Everything in Between

Learn about preferred practices for modern development

Use modern tools for a modern world

Incorporate data-centric programming for success



Brian May
Michel Mouchon
Jon Paris
Mike Pavlak
Trevor Perry
Pascal Polverini
Jim Ritchhart
Tim Rowe
Jon Rush
Paul Tuohy
Jeroen Van Lommel
Carol Woodbury
Nadir K. Amra
Hernando Bedoya
Tony Cairns
Dan Cruikshank
Rich Diedrich
John Eberhard
Mark Evans
Antonio Florez
Susan Gantner
Jesse Gorzinski
Isaac Ramirez Herrera



International Technical Support Organization

**Modernizing IBM i Applications from the Database up
to the User Interface and Everything in Between**

June 2014

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

First Edition (June 2014)

This edition applies to IBM i 6.1 and IBM i 7.1 and later.

© Copyright International Business Machines Corporation 2014. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	xi
Trademarks	xii
Preface	xiii
Authors	xiii
Now you can become a published author, too!	xix
Comments welcome	xx
Stay connected to IBM Redbooks	xx
Chapter 1. Why modernize	1
1.1 What is modernization	2
1.2 Approaches to modernization	2
1.3 The need for modernization	3
1.3.1 The need for database modernization	3
1.3.2 The need for interface modernization	3
1.3.3 The need for program modernization	4
1.4 The benefits of modernization	4
1.5 Where are you starting from	4
1.6 Laying the ground rules	5
1.7 What a modern application is	6
1.8 Roadblocks	7
1.9 The journey begins	7
Chapter 2. Path to modernization	9
2.1 Where are you starting	10
2.2 What are your modernization goals	11
2.2.1 Replacement	11
2.2.2 Reengineering	11
2.2.3 Refacing	12
2.2.4 Refactoring	14
2.3 How to modernize	14
2.3.1 Introduction	14
2.3.2 Modernization model	15
2.3.3 General principles	17
2.3.4 Phase 0: Prepare	19
2.3.5 Phase 1: Experimentation	22
2.3.6 Phase 2: Off and running	22
2.3.7 Phase 3: Evaluate and expand, future thinking	26
2.3.8 Phase 4: Deployment	27
2.3.9 Repeat as necessary	28
Chapter 3. Modern application architecture techniques	29
3.1 Introduction	30
3.1.1 Symptoms of a bad design	30
3.2 Modularization	31
3.2.1 Why modularization is important	31
3.2.2 General principles	33
3.3 Tiered design	34
3.4 Business object model	35

3.4.1	Implementation	35
3.4.2	Advantages	36
3.5	Data-centric programming	37
3.5.1	Moving to data-centric programming	40
3.6	Service orientation	41
3.6.1	What is a service	42
3.6.2	The properties of the services	43
3.7	Cloud	44
Chapter 4. Modern development tools		45
4.1	Editing, compiling, and debugging IDE	46
4.2	SEU and PDM to Rational Developer for i: Why change	46
4.2.1	Current (and future) language support	46
4.2.2	Eclipse-based development environment	47
4.2.3	Productivity features of Rational Developer for i	47
4.2.4	Moving from PDM or SEU to Rational Developer for i	53
4.3	Other tools	55
4.3.1	Application Management Toolset	55
4.3.2	RPG Next Gen	57
4.3.3	RPGUnit	58
4.3.4	Zend Studio for IBM i	58
4.4	Source control and development management tools	59
4.5	IBM Rational Team Concert and ARCAD Pack for Rational	64
4.5.1	Description	65
4.5.2	Example of IBM Rational Team Concert configuration	72
4.5.3	Managing IBM i projects with IBM Rational Team Concert	73
4.5.4	ARCAD Pack for Rational	76
Chapter 5. Interfacing		77
5.1	Stopping the confusion: Introduction to interfacing	78
5.1.1	The past	78
5.1.2	The present	78
5.2	Stateless interfaces	79
5.3	Communication	79
5.3.1	IBM WebSphere MQ	80
5.3.2	Data queues	82
5.3.3	TCP/IP	84
5.3.4	SSL/TLS	85
5.3.5	HTTP/HTTPS	86
5.3.6	FTP/FTPS	87
5.3.7	SSH/SFTP	87
5.3.8	Sockets	88
5.4	SOA and web services	92
5.4.1	Web services that are based on SOAP	94
5.4.2	Web services that are based on REST	94
5.4.3	SOAP versus REST	95
5.4.4	Web services support on IBM i	96
5.4.5	Web services: Provider perspective	96
5.4.6	Integrated web services server	97
5.4.7	WebSphere Application Server and IBM Rational Developer for i	103
5.4.8	Enterprise service bus	110
5.4.9	Integrated web services client for ILE	111
5.4.10	Web services: PHP	116

5.4.11 REST Web Services: XML Service	128
5.5 Cross environment	128
5.5.1 Cross environment: Java to ILE	128
5.5.2 Cross environment: XMLSERVICE	135
5.5.3 Data interchange	145
5.5.4 Database access	163
Chapter 6. User interface	173
6.1 Text-based user interface	174
6.2 Browser user interface	176
6.3 Rich-client user interface	180
6.4 Custom application user interface	183
6.5 Summary	186
Chapter 7. Modern RPG	187
7.1 The definition of modern RPG	188
7.2 The business value of modern RPG	189
7.3 Moving to modern RPG	190
7.4 Preferred practices	192
7.4.1 Preferred practices for RPG program design	192
7.4.2 Preferred practices for RPG coding	195
7.4.3 Quick list of RPG preferred practices guides	203
7.5 Free format RPG	204
7.5.1 Why free format	204
7.5.2 Free format RPG syntax overview	205
7.5.3 Free format examples	207
7.6 ILE and modularization	213
7.7 A modern language deserves modern tools	215
7.8 Dead code elimination	215
7.8.1 Why code “dies”	215
7.8.2 Dead code risks	216
7.8.3 Dead code detection	216
7.8.4 The iTrace tool	216
7.8.5 Using caution when removing code	218
7.9 Cyclomatic complexity and other metrics	218
7.9.1 Other relevant metrics	218
7.9.2 Putting it all together	220
Chapter 8. Data-centric development	221
8.1 Set-based thinking	222
8.2 Defining physical containers for sets (SQL DDL)	222
8.2.1 Referential integrity	222
8.2.2 Indexes	226
8.2.3 Views	232
8.2.4 Auto-generated column support	233
8.3 Accessing and operating on sets (SQL DML)	236
8.3.1 Static	236
8.3.2 Dynamic	238
8.3.3 Security	243
8.3.4 SQL data access interfaces	245
8.3.5 Preferred practices from a database perspective	255
8.3.6 Result sets	260
8.3.7 Joins	262
8.3.8 Commitment control	267

8.4 Database modularization	279
8.4.1 Stored procedures	280
8.4.2 Triggers	283
8.4.3 User-defined functions	284
8.5 Modern reporting and analysis	285
8.5.1 Removing OPNQRYF and RUNQRY	286
8.5.2 Reasons to consider conversion	287
8.5.3 SQL is the industry standard	287
8.5.4 SQE features	287
8.5.5 Superior performance of SQE	288
8.5.6 Advanced functions of the SQL interface	288
8.6 Database tools	290
8.6.1 Managing DB2 for i with IBM Data Studio	291
8.6.2 Managing DB2 for i with InfoSphere Data Architect	310
8.6.3 Managing DB2 for i with IBM i System Navigator	321
8.6.4 SYSTOOLS	328
8.6.5 XML	333
8.6.6 Using RPG to use DB2 XML support	340
Chapter 9. Database re-engineering	351
9.1 Re-engineering versus re-representing	352
9.2 Getting started	353
9.2.1 Business requirements that might result in re-engineering the database	353
9.2.2 Training, teaming, and tools	356
9.3 The phased approach to seamless transformation	358
9.3.1 Phase 0: Discovery	358
9.3.2 Phase 1: Transformation	367
9.3.3 Phase 2 and 3	392
9.3.4 Creating SQL views	392
9.3.5 Creating and deploying the SQL I/O modules	396
9.3.6 Creating and deploying external SQL I/O modules	418
9.3.7 Dynamic SQL	423
9.3.8 Bridging existing programs to SQL service programs	424
9.3.9 Activating referential integrity	427
9.4 Phase 3 and beyond	441
9.4.1 Managing database objects	442
9.4.2 Changing the DB2 for i database	443
9.4.3 Creating a table	445
9.4.4 Modifying a table by using the graphical interface of IBM i Navigator	447
9.4.5 Changing the PDM	453
9.4.6 Database update scenarios	460
Chapter 10. Mobile, UI, and database modernization tools	463
10.1 Why you should consider a modernization tool	464
10.1.1 Mobile and UI modernization tools	464
10.1.2 Why you should consider one of these solutions	466
10.2 IBM Rational Open Access: RPG Edition	467
10.2.1 Architectural modernization	471
10.2.2 New control capabilities	472
10.2.3 User interface	475
10.2.4 Flow	478
10.2.5 Debugging	479
10.2.6 Converting existing programs to Open Access	479

10.2.7	Creating programs with Open Access.	480
10.2.8	Enterprise integration	480
10.2.9	Open Access Handler example.	481
10.2.10	Conclusion	484
10.3	5250 data stream interpretation	485
10.3.1	How the 5250 data stream method works.	485
10.3.2	IBM Rational Host Access Transformation Services.	486
Chapter 11. EGL	489
11.1	A closer look at EGL	490
11.2	EGL modernization options for IBM i	492
11.2.1	Modernization through access to existing data	493
11.2.2	Modernization through reuse of existing RPG programs	494
11.3	A brief look at the EGL language	495
11.4	EGL rich user interface for web and mobile interfaces	500
11.4.1	The need for Web 2.0 and rich internet applications.	501
11.4.2	Challenges in building Web 2.0 and rich internet applications	501
11.4.3	Web 2.0 and rich internet applications with EGL	502
11.4.4	EGL rich user interface syntax and tools	502
11.5	EGL: From the database to the web	507
11.5.1	Creating a data access application and services	507
11.5.2	Creating a data access application	508
11.5.3	Creating services and testing a local web service	518
11.5.4	Creating an EGL rich user interface front end.	524
11.6	EGL: Calling external programs	536
11.6.1	Using linkage to call an RPG program	536
11.7	Benefits of EGL for IBM i.	543
Chapter 12. PHP	545
12.1	What is PHP	546
12.1.1	Open source	546
12.1.2	Documentation	546
12.1.3	What does PHP look like	547
12.1.4	How PHP works	548
12.1.5	Why use PHP	548
12.1.6	Who is Zend	550
12.1.7	PHP environment setup on IBM i	550
12.1.8	What is Zend Server	551
12.2	PHP development	552
12.2.1	Zend Studio.	552
12.2.2	Exploring the stair-step approach: Four steps	558
12.2.3	Other data access.	565
12.3	HLL integration	566
12.3.1	What the toolkit is made of	566
12.3.2	High-level description and workflow	567
12.3.3	Program calls	567
12.4	How dates are different in PHP.	574
12.4.1	Installing open source applications.	581
Chapter 13. Java	585
13.1	The world of Java	586
13.1.1	What is Java	586
13.1.2	Java on IBM i	588
13.2	Why Java	594

13.2.1	Language and platform characteristics	595
13.2.2	Tools	598
13.2.3	Documentation and community	600
13.2.4	Freedom of choice	600
13.2.5	Skills and new talent	601
13.3	Moving to Java	602
13.3.1	The learning curve	602
13.3.2	Myths surrounding Java	603
13.3.3	Further reading	604
Chapter 14.	Web serving: What and why	605
14.1	Web server versus application server	606
14.1.1	Web server	606
14.1.2	Application server	608
14.2	WebSphere Application Server	608
14.3	Integrated Web Application Server	610
14.4	Tomcat	611
14.5	PHP: Zend Technologies	612
14.6	Ruby: PowerRuby.com	612
14.7	Conclusion	612
Chapter 15.	HTML, CSS, and JavaScript	613
15.1	The development environment	614
15.2	The browser interface	614
15.3	HTML	617
15.4	CSS	618
15.5	JavaScript	620
15.6	Ajax	623
15.7	HTML5 and CSS3	623
15.7.1	HTML5	624
15.7.2	CSS3	628
15.8	The benefit of frameworks	631
15.9	Going to the web	632
Chapter 16.	Security	633
16.1	Introduction	634
16.2	Classifying data	634
16.3	User authentication	635
16.3.1	IBM i user profile	635
16.3.2	Validation list user	635
16.3.3	LDAP	636
16.3.4	Application-defined authentication	637
16.3.5	Kerberos	637
16.3.6	*ALLOBJ special authority	637
16.3.7	Application owner profile	638
16.3.8	Connection profiles	638
16.3.9	Logging	638
16.3.10	Separation of duties	639
16.3.11	Encryption	639
16.3.12	Authority settings on the data	640
16.3.13	Other *PUBLIC authority settings	641
16.3.14	Using authorization lists to secure application objects	641
16.3.15	Integrated file system considerations	641
16.3.16	Conclusion	642

Chapter 17. Globalization	643
17.1 Terminology	644
17.1.1 From an encoding scheme to a CCSID	644
17.1.2 Character set	645
17.1.3 Code page	647
17.1.4 Coded character set identifier	649
17.1.5 Invariant character set	650
17.1.6 Unicode	651
17.2 Single-, double-, and multi-byte character set (SBCS, DBCS, and MBCS)	651
17.2.1 DBCS and MBCS data type	652
17.3 Data integrity between environments or languages	652
17.3.1 Environments	652
17.3.2 Languages	652
17.3.3 Why a CCSID is important	653
17.3.4 Does a program have a CCSID	653
17.3.5 Flow examples	654
17.4 Unicode	656
17.4.1 Unicode CCSID	656
17.4.2 Unicode examples from the DB through the RPG to the DSPF	657
17.4.3 Moving to Unicode	660
17.4.4 Emulators	661
17.5 Locales	661
17.6 Reference links	662
Chapter 18. Cloud	663
18.1 Overview: IBM i and cloud	664
18.2 IBM i cloud considerations	664
18.2.1 Security and isolation	664
18.2.2 New tenant provisioning	664
18.2.3 Service-level agreements	665
18.2.4 Licensing and pricing	665
18.2.5 Deployment options	665
18.3 IBM i OS cloud and multitenancy	666
18.3.1 Physical/isolated multitenancy	667
18.3.2 Shared hardware multitenancy	667
18.3.3 Operating system level multitenancy	667
18.3.4 Platform-level multitenancy	668
18.3.5 Application-level multitenancy	668
18.4 IBM i cloud technical refresh capabilities	669
18.4.1 Virtualization	669
18.4.2 Virtualization technologies for cloud and IBM i	669
18.5 IBM i virtual appliances	670
18.6 Summary	671
Related publications	673
IBM Redbooks	673
Other publications	673
Online resources	674
Help from IBM	675

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 grant 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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

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

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

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


COPYRIGHT LICENSE:

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

Trademarks

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

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

Active Memory™	IBM®	PowerVM®
AIX®	InfoSphere®	POWER®
AS/400®	Integrated Language Environment®	PureFlex®
DB2 Connect™	iSeries®	PureSystems®
DB2 Universal Database™	Jazz™	Rational Team Concert™
DB2®	Language Environment®	Rational®
developerWorks®	Lotus Notes®	Redbooks®
Distributed Relational Database Architecture™	Lotus®	Redbooks (logo)  ®
DRDA®	Net.Data®	RPG/400®
eServer™	Notes®	System i®
i5/OS™	OS/390®	Systems Director VMControl™
IBM Flex System Manager™	OS/400®	Tivoli®
IBM Flex System®	Power Systems™	WebSphere®
IBM SmartCloud®	PowerHA®	z/OS®
	PowerPC®	

The following terms are trademarks of other companies:

Adobe, the Adobe logo, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

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

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

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

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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

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

Preface

This IBM® Redbooks® publication is focused on melding industry preferred practices with the unique needs of the IBM i community and providing a holistic view of modernization. This book covers key trends for application structure, user interface, data access, and the database.

Modernization is a broad term when applied to applications. It is more than a single event. It is a sequence of actions. But even more, it is a process of rethinking how to approach the creation and maintenance of applications. There are tangible deliveries when it comes to modernization, the most notable being a modern user interface (UI), such as a web browser or being able to access applications from a mobile device. The UI, however, is only the beginning. There are many more aspects to modernization.

Using modern tools and methodologies can significantly improve productivity and reduce long-term cost while positioning applications for the next decade. It is time to put the past away. Tools and methodologies have undergone significant transformation, improving functionality, usability, and productivity. This is true of the plethora of IBM tools and the wealth of tools available from many Independent Solution Providers (ISVs).

This publication is the result of work that was done by IBM, industry experts, and by representatives from many of the ISV Tool Providers. Some of their tools are referenced in the book. In addition to reviewing technologies based on context, there is an explanation of why modernization is important and a description of the business benefits of investing in modernization. This critical information is key for line-of-business executives who want to understand the benefits of a modernization project. This book is appropriate for CIOs, architects, developers, and business leaders.

Authors

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



Nadir K. Amra has worked on web technologies on IBM i for over 15 years, and is the lead architect and developer for integrated web services support on IBM i.



Hernando Bedoya is a Senior IT Specialist at STG Lab Services and Training, in Rochester, Minnesota. He writes extensively and teaches IBM classes worldwide in all areas of DB2® for i. Before joining STG Lab Services, he worked in the ITSO for nine years writing multiple IBM Redbooks publications. He also worked for IBM Colombia as an IBM AS/400® IT Specialist doing presales support for the Andean countries. He has 28 years of experience in the computing field, and has taught database classes at Colombian universities. His areas of expertise are database technology, performance, and data warehousing. He holds a Master's degree in Computer Science from EAFIT, Colombia.

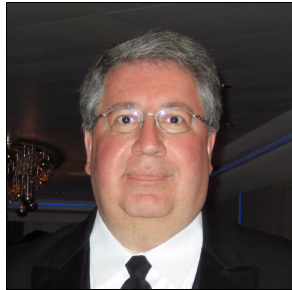


Tony Cairns is a Senior Programmer for IBM in Rochester, Minnesota. His career includes many positions ranging from management to technical. He is a member of the IBM i5/OS™ PASE/PHP/Ruby team. You can email Tony at adc@us.ibm.com.



Dan Cruikshank has been an IT Professional since 1972. He has consulted on many different project areas since joining IBM Rochester in 1988. Since 1993, Dan focused primarily on resolving IBM System i® application and database performance issues at several IBM customer accounts. Since 1998, he is one of the primary instructors for the Database Optimization Workshop. Most recently, he is a member of the DB2 for i Center of Excellence team with IBM Rochester Lab Services.

Dan has become proficient in several application development languages and is recognized as a subject matter expert (SME) for re-engineering DB2 for i databases. In addition, he is proficient in the usage of the following modern IBM application development tools: IBM Data Studio, IBM Rational® Developer for i, and InfoSphere® Data Architect. He is in strong demand worldwide and is frequently brought in to assist with executive briefings, customized education, technical conferences, user group presentations, and the occasional proof of concept. Dan has published several articles relating to IBM i performance analysis, database optimization, and database re-engineering. More recently, he has done webinars and user group presentations on application development methodologies and database re-engineering topics (for example, data description specification (DDS) to Data Definition Language (DDL) and IBM Rational Developer for i). As an IBM Senior Consultant, his main role is to illuminate, educate, and enable IBM i customers in how to take advantage of this remarkable system and its integrated, state-of-the-art relational database.



Rich Diedrich is an IBM Senior Technical Staff Member and IBM Master Inventor with over 32 years experience working at IBM Rochester in various areas. His most recent focus is on application modernization and globalization, particularly modularization of traditional IBM i languages (RPG and COBOL).



John Eberhard is a Software Engineer at IBM Rochester. He has worked with AS/400 and IBM i for over 20 years. He currently works on the IBM i Data Access team, where he supports the native and toolbox JDBC drivers, and the OLEDB and .NET providers.



Mark Evans is the IBM Rational Business Developer Product Manager. He is responsible for setting product direction and working with customers who use the EGL technology within the Rational Business Developer product for application development, which includes IBM i and the other platforms and technologies for which EGL can be used. Before becoming the product manager, for 25 years he worked in the lab or lab services teams on EGL and its predecessor technologies in the areas of application deployment, application development preferred practices, product integration issues, and other runtime-related considerations across all platforms, including IBM i.



Antonio Florez is an IT Specialist at IBM, in Colombia. Before joining IBM in Colombia more than four years ago, he worked for 16 years in software development and IBM i consulting for many clients in Colombia and South America. He has 20 years experience in the computing field and has taught DB2 for i database, RPG, and development tools classes in Colombian universities. He is a technical leader for two projects for a large client in Colombia. His areas of expertise are database technology and application development in RPG and Java.



Susan Gantner began her career as a Programmer for companies in Atlanta on various platforms. After working with her first IBM System/38, she vowed to stick with that platform. The System/38 became the IBM AS/400, since renamed a few times to what we now know as IBM i.

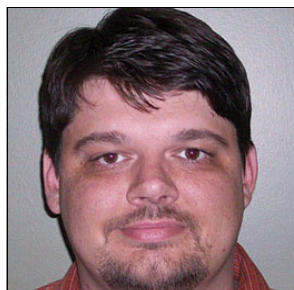
After application programming for 10 years outside IBM, she joined IBM as a Systems Engineer and later worked in both the IBM Rochester and IBM Toronto labs, supporting developers on her favorite platform. Today, post IBM, she works primarily as an educator teaching modern application development techniques through onsite classes that are customized for specific companies' needs and at technical conferences, such as the RPG & IBM DB2 Summit. She has authored many articles on various topics that are relevant to IBM i developers.



Jesse Gorzinski is a software engineer for IBM. He has worked for IBM since 2006 and has contributed to several key IBM i deliverables, including IBM Technology for Java (J9) Java virtual machine and IBM i Access Client Solutions. Before his employment at IBM, Jesse worked as an IT administrator for an IBM customer in the finance and mortgage industry, where he developed a fondness for IBM i. There, he was responsible for data backup and recovery, process optimization, information integrity, and modernization.



Isaac Ramírez Herrera is the modernization team lead for BAC Credomatic Network in Costa Rica. Passionate about everything that is related to software development, he especially enjoys research, development, and teaching tasks. Working on the IBM i platform since 2004, Isaac focuses on applying his broad range of knowledge in many programming languages and technologies to design and build modern applications. He has a degree in computer engineering and is finishing a Master's degree in Information Systems from the Instituto Tecnológico de Costa Rica.



Brian May is an IBM i Modernization Specialist at Profound Logic Software. He divides his time between product development, customer projects, and evangelizing RPG and IBM i. He began his IBM i career in 2001 as an RPG developer and has since branched out as a writer, speaker, and educator on topics that are related to application development with RPG, HTML, JavaScript, and PHP. Because of his efforts, IBM recognized him as one of the initial IBM Champions for Power Systems™ in 2011.



Michel Mouchon has held the position of CTO at ARCAD Software since 2000, playing a major role in the definition of technical strategy and the coordination of R&D, Engineering, and Consultancy departments. He has been instrumental in steering ARCAD Software towards its current position of technology leader in Application Lifecycle Management (ALM), maintaining a constant focus on modernization of applications, methods, and tools. His commitment to modernization on the IBM i platform led to the integration of the ARCAD Pack for IBM Rational within the IBM Rational offering for Collaborative Lifecycle Management (CLM), which launched in 2011. This same solution won ARCAD Software first prize in Enterprise Modernization at IBM Innovate in both 2012 and 2013 and is now distributed by IBM Rational worldwide.

Michel has built highly transversal skills in IT, starting his career with a dual degree in electronics and software engineering, working as Analyst Programmer on IBM mini-systems, appointed senior consultant, and Systems and Methodology Manager with an advisory role on IBM technology in audits, development, and training.



Jon Paris has spent most of his 45 years in the IT industry working with midrange systems. His experience covers the spectrum from operations to programming to management. In the early 1980s, Jon was introduced to the IBM System/38 and fell in love. Eventually that love affair led to him joining the IBM Toronto Lab where he worked on the COBOL and RPG compilers for the AS/400. He was also with the initial team that designed the RPG IV compiler.

Since leaving IBM, Jon has spent his time helping customers to modernize their applications and development practices. He does this through custom training classes, magazine articles, conference education, web casts, and as one of the authors of *Who Knew You Could Do That with RPG IV? A Sorcerer's Guide to System Access and More*, SG24-5402.



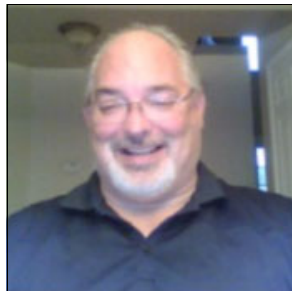
Mike Pavlak has been working with IBM midrange solutions since 1992 and IBM Mainframes before that. After years of developing applications using RPG, CL, and PHP, he managed IT development and IT for power protection manufacturer Tripp Lite. Mike joined Zend Technologies as a Solutions Consultant working with the IBM sales team in 2008 and enjoys helping IBM i customers explore PHP as a state of the art solution for application modernization. In addition to several roles as a volunteer with COMMON, the premier IBM midrange user group, Mike also teaches PHP classes part time at Moraine Valley Community College in suburban Chicago.



Trevor Perry is a speaker and consultant on modernization and IT strategy around the globe and occasionally at home in New York. His sessions on a united approach to IT and business strategies challenge traditional methods and lead companies as they move their IT organizations into the future.



Pascal Polverini is part of the IBM i ISV Advisory Council, a pioneer in RPG Open Access, and a driving force of the RPG OA Metadata Open Standard, providing modernization solutions for business-critical RPG applications.



Jim Ritchhart has a career that has spanned more than 25 years of application development and system design. He began as a programmer developing applications for Baxter Healthcare and Handy Andy Home Improvement Center, then consulted for Kraft Foods and Londen Insurance Company before joining Uline. Jim routinely speaks at conferences and user groups about DB Modernization, Data Centric Application Architecture, SQL Performance Tuning, and RPG Application Development. Jim consults for various companies assisting with SQL Performance Tuning, Application Design and Development, and setting up data communications between DB2 for IBM iSeries® and other disparate DBMS.



Tim Rowe is the Business Architect for Application Development and Systems Management for IBM i. He is responsible for ensuring that the IBM i operating system has the features, function, languages, and tools IBM i developers require to run, maintain, and write modern applications to run business. He has been working in the web and IBM i middleware space for the past decade. He regularly speaks at conferences to help customers understand how IBM i is a modern platform for running the most modern applications. He has been a part of the IBM i family for over 20 years and has worked on many different layers of the operating system.

Jon Rush is a certified IT consultant in IBM Power Lab Services specializing in IBM i virtualization on Power and IBM PureFlex® hardware, IBM i SaaS, MSP and cloud deployment, and IBM WebSphere®, Java, and PHP technologies on IBM i.



Paul Tuohy is one of the most prominent consultants and trainer/educators for application modernization and development technologies on the IBM Midrange. He currently holds positions as CEO of ComCon, a consultancy firm that is based in Dublin, Ireland, and is a founding partner of IBM System i Developer, the consortium of top educators who produce the acclaimed RPG and DB2 Summit conference. Previously, he worked as IT Manager for Kodak Ireland Ltd. and Technical Director of Precision Software Ltd.

In addition to hosting and speaking at the RPG and DB2 Summit, Paul is an award-winning speaker at COMMON and COMMON Europe Congress.



Jeroen Van Lommel is Competence Center Coordinator for System i at ASIST, a Belgian IBM Business Partner. He leads from a business and organizational perspective. Within this perspective, he focuses on modernizing IBM i architecture to a modern platform for IT departments to maintain and for users to work with.

He holds a Bachelor's degree in Applied Computer Science and SME Management. He also holds various IBM certifications, including IBM i 6.1, IBM i 7, and IBM WebSphere Application Server V8.0.



Carol Woodbury is President and co-founder of SkyView Partners, a firm specializing in managed security services and security administration, and compliance software and services. Carol has been working in the area of security since 1990, including 10 years as the AS/400 Security Architect and Chief Engineering Manager of Security Technology in Rochester, MN. Carol is Certified in Risk and Information Security Control (CRISC).

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 customer 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/IBMRedbooks>

- ▶ 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>



Why modernize

For the IBM i community, the need to modernize applications is now an urgent requirement. The world of business applications has become a fast-moving, ever-changing environment. The web, mobile computing, and the cloud have quickly followed the impact of graphical interfaces. The community is facing an ever-changing environment and it must ensure that applications can be easily enhanced and maintained when dealing with these new requirements.

The choices for modernizing an application are either to replace it or to re-engineer it. As to which of these two options is better depends on the application and why the modernization process is being undertaken. No matter which option is the correct solution, there is still a requirement to understand the modernization process and the re-engineering process that is adopted or rejected.

Traditional applications are often large and difficult to modify. Modernizing an application does not mean that the application must be replaced or rewritten in a new language. Modernization means re-engineering an application. This might include changing any combination of the interface, business logic, and database. The goal of modernization is to make it easier to change and maintain your application in the future and provide the correct interface for your users.

The reality is that, over the years, companies already have made a heavy investment in applications that satisfy business needs. Although business requirements might be changing, the current business needs must be satisfied. The challenge is to keep and reuse those core business processes while reengineering, refactoring, and enhancing the rest of the application to meet these demands.

This chapter describes the driving forces behind modernization and some of the commonalities behind modernization projects. It covers the following topics:

- ▶ What is modernization
- ▶ The need for modernization
- ▶ The benefits of modernization

Tip: The only mistake is to do nothing.

1.1 What is modernization

Modernization means different things to different people. Modernization might mean one or more of the following items:

- ▶ Creating a user interface
- ▶ Refactoring a database
- ▶ Making major changes to code
- ▶ Making minor changes to code
- ▶ Making no changes to code
- ▶ Integrating new applications
- ▶ Any permutation or combination of the preceding items

Modernization can be anything from screen scraping an application to converting it to a tiered or service-oriented architecture (SOA).

Modernization can consist of any permutation and combination of using an independent software vendor (ISV) solution, redesigning a database, redesigning and recoding application programs, coding new programs, using new programming languages, and writing new applications. Modernization is a broad term with many different aspects to consider. This goal of this publication is to provide a holistic view of modernization.

1.2 Approaches to modernization

The developer's job is much easier if there is one simple roadmap for modernization. Unfortunately, such is not the case.

Many factors determine the wanted approach to modernization:

- ▶ Budget.
- ▶ Management support.
- ▶ Why the modernization process is being undertaken. What permutation of interface, database, and application code must be modernized?
- ▶ Resource. The challenges that are faced by a hundred-developer shop are different from those faced by a ten-developer shop, which are, in turn, different from those faced by a two-developer shop.
- ▶ What is the state of the current application? How easy is it going to be to change that code?

Any one of these items can be broken into multiple bullet points.

Because there are different requirements, different starting points, different levels of support, and different levels of resource, there are many different approaches to modernization.

Although there are many approaches, one of the key challenges is to ensure that the chosen approach is fluid. One must walk the line between control and flexibility and develop the ability to embrace what is new while maintaining a working core.

At the outset, choosing a modernization path seems like a daunting task. If one accepts that the path changes when the process starts, it is much easier to get started.

1.3 The need for modernization

The need for modernization is prompted by a business need; this often means that users prompt the need for modernization. Although you might know that, from a technical point of view, an application should be modernized, it is rare that you have the resources and necessary financing without the requests and support of users and management.

Users are now more technologically savvy. User demand is not just for solutions. Users are also demanding how they want the solutions delivered.

Business demands lead to modernization requirements in three areas:

- ▶ Database
- ▶ Interface
- ▶ Application code

Although there are many variations, the following examples should sound familiar.

1.3.1 The need for database modernization

Here are some common scenarios that can lead to database modernization:

- ▶ The company decides to replace Query/400 with IBM DB2 Web Query.
- ▶ The database must be modernized to make it more accessible to users and to ensure that the use of software such as DB2 Web Query does not adversely affect the performance of the system.
- ▶ Users start to use a new PC analysis tool that extracts information from the database. The tool vendor recommends replicating the data onto another platform because it does not look like a database to them. The database must be refactored to represent it in a format more familiar to a database administrator (DBA).
- ▶ A new application is written over an existing database. The new application uses database features (for example, constraints and triggers) that are not implemented in the existing database. Implementing these features might also affect the existing application.
- ▶ An existing application is rewritten. Modernizing the database reduces the complexity of the programs in the new application.

1.3.2 The need for interface modernization

The need for new or enhanced interfaces is probably the most often quoted reason for modernization. These are some common scenarios that can lead to interface modernization:

- ▶ A company invests in new applications that use a graphical user interface (GUI). Users are switching between GUI and traditional 5250 interfaces for different applications. They prefer a common interface, and a GUI is their preferred interface.
- ▶ There are now generations of users who have never encountered a 5250-style screen before joining the company. To these users, a GUI is the interface of choice and a 5250-style screen is considered old-fashioned.
- ▶ Users who have been using an application for a number of years request that a new GUI interface be put in to place, often because of poor screen design. The users are happy with the application, but not with the way it looks. Often, when users are happy with the 5250-style design, they do not want a new interface until new users who have not experienced the 5250-style design come along.
- ▶ The CEO wants monthly reports as a single graph on their smartphone.

- ▶ Perception. 5250 is considered old and GUI or mobile is considered new. New applications must be better, right? Often, a new UI can dramatically change the community view.
- ▶ Which generation of users need instructions about how to use a mouse?

1.3.3 The need for program modernization

Here are some common scenarios that can lead to program modernization:

- ▶ A new web-based interface that replicates an existing 5250-style application is being developed. For example, an order entry application is being created so customers can place their own orders without having to call them in. The new application must use the same business logic as the existing application. Unfortunately, the business logic is firmly embedded in the 5250 screen programs.
- ▶ Because of changes that are being made to the database or the interface, changes must be made to the application programs.
- ▶ An application has four horrendous, gigantic programs that contain the core business logic of the application. Any change (even a minor one) to one of these programs is a massive undertaking that requires a detailed knowledge of the programs and extensive testing. It is time to simplify your maintenance cost.
- ▶ The programmers with the detailed knowledge of the application programs are nearing retirement. It is better if the code is in a format that is easier for a recently trained programmer to understand. It is time to use modern programming styles and tools.
- ▶ Customers must access existing information through a web service as opposed to existing interfaces.

1.4 The benefits of modernization

There are many benefits to be gained from modernization:

- ▶ A better interface
- ▶ A better database
- ▶ Easier to maintain applications
- ▶ More flexible applications. New business requirements are easier to implement.
- ▶ Integrated applications. It is easier to integrate with other applications, platforms, and interfaces.
- ▶ It is easier to find developers who can maintain modern code.
- ▶ A modernized application can give a company a competitive edge. The ability to make rapid changes to the application means that the company can respond quickly to business changes.
- ▶ Continue to use the existing investment. The fact that an application might benefit from an update to its interface does not mean that the application is not working and fulfilling a business need.

1.5 Where are you starting from

As with all journeys, the length of the journey and the challenge of the terrain to be traversed depends on where you start.

The advantage and disadvantage of IBM i is that applications are compatible with earlier versions. It is possible for a program that was written on the System/38 in 1980 to run (with no recompilation) on one of today's IBM Power Systems. The benefit is that, as hardware was replaced and improved, applications did not need to be upgraded or changed. Unfortunately, the same applications are written using code and coding techniques that might have been leading edge twenty plus years ago but fall short of today's requirements.

You need to determine the following items:

- ▶ Does the application have a well-structured relational database?
- ▶ Was the application originally written for System/34, System/36, System/38, or one of the many flavors of IBM AS/400, IBM eServer™ systems, IBM iSeries, or IBM i?
- ▶ What language is the application written in? Is it COBOL, COBOL ILE, RPG II, RPG III, RPG IV, Java, C, some other language, or some permutation or combination of two or more languages?
- ▶ Is the application using the IBM Integrated Language Environment® (ILE)?
- ▶ Are the application programs well-structured?

1.6 Laying the ground rules

Starting a modernization project can be a daunting prospect and it is easy to be overwhelmed by the options and amount of information that is available. This is the stage at which most people falter: it is easier to stick with what you know as opposed to embracing what is new.

There is also the challenge of having to maintain the existing application while it is being modernized.

The scope of a modernization project, the importance of different options, and the structure of a project depend on the resources that are available and the goal for the result. There are enormous differences between the choices that are available to a single-developer shop and a hundred-developer shop. Usually, you must be selective about what is modernized and how modernization is achieved.

There might be more benefit from modernizing code as opposed to starting with the database or vice versa. Choices about the usage of new languages depend on the availability of training or programmers with the required skills. The need for a defined change management process varies greatly depending on how many programmers are working on the project. There many terms and conditions that apply.

It can be helpful if there is a single plan that can be applied to all modernization projects, but such is not the case. The process is different for every shop and every application.

Although every modernization project is different, there are many ground rules and circumstances that are common to all of them.

Here are the basic principles to apply to all modernization projects:

- ▶ Do not be afraid to fail.
- ▶ There is a need for education and training.
- ▶ Developers must use the correct tools and they must learn how to use them.
- ▶ There is a need for change management, which is change management in a development environment as opposed to a maintenance environment.
- ▶ Try an unofficial project. Go through the modernization process with just a couple of programs. Perform a proof of concept (PoC).
- ▶ Take small steps. There is much to learn.
- ▶ Put a documentation process in place. Use wikis.
- ▶ Determine the rules and disciplines for the modernization process. At the beginning of the process, the rules and disciplines are guidelines, which evolve into firmer processes.
- ▶ Do not adhere to the same practices you have been using for years. Use agile development principles to deliver smaller projects that build on each other.
- ▶ You need to do it right the first time. Otherwise, it can add to the expense.
- ▶ Consider a “tactical” solution that can build upon your strategic goal.

1.7 What a modern application is

A modern application is one where modifications or enhancements to the application have a minimum effect. For example, changes to the interface should not require changes to the business logic.

The nature of a modern application is that the design of the database and programs is such that the requirement for testing is minimized. This approach allows for rapid development and easier, less complex maintenance.

Here are some of the features to look for in a modern application:

- ▶ A tiered design that separates the application into separate components: interface, business logic, and database.
- ▶ A well-designed and tuned database that implements integrity.
- ▶ Code flexibility, which allows for a flexible means of developing new interfaces and changing business processes.
- ▶ Code reuse, which means that the same procedures are used everywhere. It does not mean that code is copied and pasted to multiple programs.
- ▶ Code maintainability, which means that there is no duplicate code and that the structure of the programs is simpler and easier to understand. Maintenance of these programs has fewer potential side effects.

1.8 Roadblocks

Apart from the obvious requirements for financing and resources, there are many conditions that can inhibit the modernization process. Based on the experience of the team, here are some of the common roadblocks that can hinder, if not derail, the modernization process:

- ▶ Developers do not use the correct tools. Developers are comfortable with what they know. Sometimes, it is easier to stay within the comfort zone of using familiar tools as opposed to facing the chore of learning how to use new tools.
- ▶ Developers do not know what they do not know. Developers who have no experience with certain technologies or techniques make decisions about how they are used, which is why a PoC is so important.
- ▶ Unlearning old habits. Many developers have been observing the same practices for many years. It can be a challenge to change these habits and embrace new ones.
- ▶ Not getting adequate training. It is not enough to install a tool and say “start using it” or to instruct developers that they should use a new technique when they do not really understand why they should use it or how to use it properly.
- ▶ There is so much to learn that developers feel overwhelmed before they start. Everything must be done in small steps.
- ▶ The whole process of modernization can be frightening. At the outset, there are so many unknowns that developers want to retreat to what they know best.
- ▶ Setting unrealistic targets. Targets are often set without understanding what is required to achieve them.
- ▶ Lack of management support. This is why it is important to start small. The sooner you can demonstrate gains, the easier it is to get management support.

1.9 The journey begins

Whichever of the many modernization paths on which you are about to set out, the journey is full of challenges, achievements, and interesting diversions. It is a journey that constantly offers new horizons.

Tip: Modernization is a journey, not a destination.



Path to modernization

This chapter provides a modernization path that you can follow regardless of the modernization approach that you choose. It also provides guidelines to help define where you should start your modernization path and the definitions of some of the most common modernization approaches that are available.

2.1 Where are you starting

Before you start a modernization effort, it is important to do an honest evaluation of your software assets to define your strategy. Even if your software is working, it might need to be modernized. There are important software quality attributes to consider beyond the functionality, such as maintainability, flexibility, usability, and performance. For this evaluation, consider the following steps:

1. Build an inventory of your core applications:

- Business purpose

Define the business purpose of the application from a business perspective. Understand which business processes are supported by the application and how critical each one is.

- Primary stakeholder

Determine who the stakeholders are that you should consider in an eventual modernization project. This list reminds you how important this project is for different people, not just the IT experts.

- Technical expert

You must clearly understand who the technical personnel are that maintain the applications. These experts know many small details about the application that can be critical in the project.

2. Evaluate each application against the following criteria:

- User interface flexibility
- Maintainability
- Performance
- Ease of integration
- Available skills
- Business value

3. Define your modernization goals:

- Better maintainability
- New user interface
- Improved code readability
- Better database access methods

4. Prioritize the inventory according to the goals.

After you complete these steps, you can start working. The modernization activities involve different aspects of the application:

- ▶ Database
- ▶ User interface
- ▶ Integration with other systems
- ▶ Business logic

You are just starting, so do not attempt to plan the entire modernization project now. Your modernization plans change over time and are refined along the way. Find a small piece that you can start with. Focus on small pieces that over time can lead to significant things.

2.2 What are your modernization goals

Depending on your modernization goals, there are several approaches that are available. What you must keep in mind is that there is not a “one size fits all” approach to the modernization process. This section introduces some of the most common approaches in application modernization, which are easily adaptable with most modernization goals.

2.2.1 Replacement

In this modernization approach, the original system is replaced by a new system. The new system can be custom-made or a commercial, off-the-shelf product. This technique can be appropriate if the modernization costs are high, if a modernization project is under a different approach, or if the existing application cannot meet the business objectives.

There can be many risks in this approach that you should consider. For example:

- ▶ There is no guarantee that the new system contains the same functions or performance as the old system, which might have a long history of changes to make it customized for the business needs. This situation probably is going to cause some degradation of the service at some level.
- ▶ The IT personnel that are involved in maintenance of the old system need training to learn about the new system. This training is definitely not optional.
- ▶ New applications can change your backup and recovery methodologies and must be considered.

2.2.2 Reengineering

Reengineering is the modernization approach that transforms the original application into a new form to improve its original characteristics, such as functionality, performance, flexibility, UI, and maintainability with less risk than replacement.

The main difference between replacement and reengineering is that, in the latter approach, you take into account the existing application as a constraint of the steps to be taken in the modernization process. You cannot ignore the old application. Usually, the reengineering approach is a two-phase process:

- ▶ Reverse engineering
- ▶ Forward engineering

Reverse engineering

Reverse engineering is the extraction of the high-level processes and structure of the original code into an easily understandable representation. The understanding of existing code that is gained in this phase is vital to the reengineering process. Given the complexity of many of the applications that must be modernized and the lack of existing functional documentation, this process can be difficult. So, there are many tools that can help you in this process.

Here are the key steps that take place in this phase:

1. Generate a structural representation of the code that helps you easily understand it.
2. Start mapping the structural representation of the code to business functions, which helps you identify reusable business rules that can be extracted for modularity.
3. Construct an architectural representation of the application so you can start understanding the application as a whole.

Reverse engineering tasks are repeated several times during the modernization process. Do not worry if you are finding it hard to understand the application. As the modernization process advances, your comprehension of the original system increases.

Forward engineering

Forward engineering, also known as “refinement”, is the creation of the new system from the high-level representation of the original code.

Here are the key steps that take place in this phase:

1. Construct an architectural representation of the wanted application.
2. Define the functional transformation from the original to the new application.
3. Write the new code.

In this process, you should apply preferred practices in terms of design, code, technologies, and tools. Later in this book, you find many tools, techniques, and practices that you should consider while constructing the new application. There are many tools to help with existing program understanding along with tools to help you write code in a modern manner.

2.2.3 Refactoring

Refactoring is the reengineering of only the user interface of the application. The main objective of this approach is to improve usability and flexibility from the user point of view without having to restructure the underlying code of the application. Refactoring is a technology that has been available for a long time. Some solutions are better than others. The one significant advantage of a refactoring solution is time to market. Using a refactoring technology, you can potentially move a 5250 “green screen” application to a web or mobile application in a matter of several days. This does not mean that you must stop at refactoring. This approach can be used as a stepping stone or first step approach. This method can give you valuable time if you are considering a reengineering approach.

Types of refactoring

There is more than one way to refactor an application. Depending on the degree to which they are going to be applied, some refactoring techniques are only superficial and do not require changes in the source code. Other techniques involve minimal changes in the code that handles the UI, but without going into the business logic or database access.

► Screen scraping

This type of refactoring focuses on converting traditional 5250 terminal-based applications into window-based interfaces, web interfaces, or mobile based interfaces. Screen scraping usually works by emulating a user on the terminal. While they are connected, they emulate keystrokes, process the display output, extract the data, and generate a new window. The user sees the new UI while the screen scraping does the intermediation. These technologies are targeted at reading the 5250 data stream and modifying what the user sees based on some merged content that can change or control layout, color, and much more. This is a great approach for applications where you no longer have access to the source code.

- ▶ Open Access (OA) based UI refacing

For interactive programs, you can also use OA to transform your application. The level in which OA acts is different from a screen scraper. The 5250 datastream is bypassed and the I/O buffer of the RPG program is directly assigned to the OA handler. The I/O buffer can be processed by field name or through a data structure for each format. This offers more flexibility and possibilities over the control and the extensions of your refacing approach. You can find more about OA in 10.2, “IBM Rational Open Access: RPG Edition” on page 467. Using the OA solution enables you to potentially control the UI in a more direct manner from your existing RPG code and pass new data directly from your new UI to your back-end RPG code.

Benefits of refacing

As with every modernization approach, refacing can bring you many benefits in the short term, including the following ones:

- ▶ Quick start.

With this approach, you can start and finish modernizing quickly. You do not have to understand the code or change the business logic, which can be helpful when finding executive support for more detailed modernization projects, such as complete refactoring.

- ▶ Focus only on the UI.

With refacing, you do not need to understand the underlying code of the application. If you use a screen scraping solution, the original application can be just a black box for you. If you go with OA, you change only the display file specification (a keyword in the F-spec). This can reduce the risk of breaking parts of the application that you do not understand.

- ▶ Start moving to other modernization approaches.

Soon you realize that you must start changing other aspects of the application to make it more maintainable. The OA approach opens many possibilities to consider when you are ready to start an integrated modernization effort. Often, OA can be a starting point to a more comprehensive modernization effort.

- ▶ Show great improvement quickly.

Risks

You have learned the benefits of the refacing approach, but there are many risks that are associated with this approach. You can give a “facelift” to the application and make it look modern to the users. However, the application is still the same inflexible and hard-to-maintain piece of software.

From a performance point of view, the application performance might be affected because it was not originally designed with the new user interface in mind. To perceive real benefits, the UI modernization must be accompanied with a redesign of the underlying code, where you can optimize it accordingly.

Refacing an application does not take into account the design of the UI. The flow and how a user might want to interact with an interface from a mobile or web perspective might be different from how the application is interacted with by using the green screen flow. For example, using a green screen approach might require you to select a menu to see more data, and on the web UI a simple select or drop-down menu might be a much better approach. Understanding the UI flow and interaction requires you to continue down the modernization path.

Refacing is more likely to be a temporary or tactical solution. Remember to clarify this situation with management because from their perspective they might think that the application is modernized and that it does not require extra effort.

2.2.4 Refactoring

Refactoring is a variation of reengineering that restructures a code base, improving its internal structure without adverse effects to its external functionality. Refactoring takes a “do no harm” approach, meaning among other things that applications interacting with the modernized system do not have to be changed at all.

The “do no harm” philosophy is ensured through extensive testing during the modernization process. Tests should be automated as much as possible. Ensure that the tests cover all aspects of the application.

Without a full set of tests, the risks of the refactoring increase because you cannot ensure that the new system has the exact external behavior as the original system and that you are not introducing bugs to the system.

When to use the refactoring approach

Refactoring is the preferred approach when your modernization goals include the following items:

- ▶ Reducing maintenance costs
- ▶ Increasing the clarity of the code
- ▶ Revitalizing the application and enabling it to adapt better to business needs
- ▶ Reducing the risk of affecting others by modernizing an application
- ▶ Keeping the application in the same platform and language as the original
- ▶ Using the existing skills of your current programming force
- ▶ Allowing user interface flexibility (It is easier to update the application business logic to be accessed by either different or multiple UI technologies.)

Some refactoring is often necessary regardless of the modernization approach that you choose. This book describes a modernization model that guides the modernization project. Some of the activities that are described in this model are more related to the refactoring approach. You must evaluate whether your project needs these activities.

2.3 How to modernize

This section provides general principles that must be considered in any modernization effort, regardless of the approach that you choose. It also provides a model to guide you through a modernization project. Following a model and a set of predefined rules helps you to achieve a mature modernization process. You should adapt this model to meet your needs.

2.3.1 Introduction

Modernization is a discovery process and you must learn a lot. But even so, it is important to follow a model that can guide your way. There will be more than just one modernization project ahead, so prepare to record your journey so others can repeat your success and avoid your mistakes.

This section provides you with a model that can help you understand what you must do to modernize an application. There are some general principles that are applicable not only to this model but to any custom process that you define.

Modernization is not constrained to specific languages or platforms. There are thousands of software applications that must be modernized, and you are not the only one who has needed to undertake a modernization project. These principles and this model are applicable no matter what your situation.

The model that you are about to learn is not “carved in stone”. You should take the parts that are applicable to your modernization approach and your specific situation and adapt them.

There is plenty of room for improvement and creativity. Adopt the model and improve it any way that you need. Make sure that others can repeat it and improve it themselves so that you have a mature process that can reduce risks and uncertainty.

2.3.2 Modernization model

This section presents the modernization model that is proposed in this book. You are introduced to the principles that guide the modernization model and a prescriptive set of phases that you can follow to achieve your modernization goals.

A model or a methodology

Before you learn about the modernization model, it is necessary to have a clear understanding of what is a *model* and what is a *methodology*. Sometimes these terms are mistakenly used as synonyms. A model defines an approach and is a scheme to do things. A model defines what you need to do, not how to do it. On the other side, a methodology defines every activity that must be done to achieve every step of the model. A methodology can be thought of as the implementation of a model.

This book defines a modernization model instead of a methodology. Modernization can be approached in different ways and it might be hard to track every new technique, but a model can be time-proof. It is your responsibility to adapt this model to your specific situation, keeping in mind the principles, guidelines, and phases that every modernization effort should have.

The model

Modernizing applications is more than just following steps and performing some activities. It implies a set of principles that you must remember. The modernization model that is proposed defines this set of principles and also a set of phases to achieve your modernization goals.

The modernization model is shown in Figure 2-1. As you can see, the model defines an iterative modernization cycle that is repeated as many times as you need. Every iteration contributes to the main objective: build a modern application.

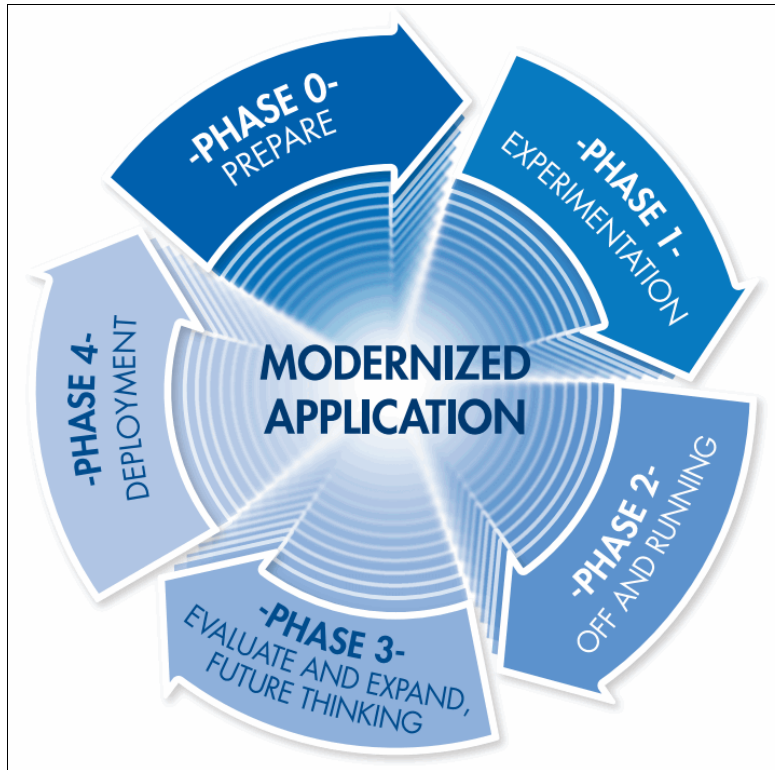


Figure 2-1 Modernization model

After every iteration, you should stop and reassess the situation. You can change your modernization approach or any specific technology that you selected. The key point is that you can see results early and you can stop as soon as your modernization goals are achieved.

In the following section, you see the definition of the phases that compose the model:

- ▶ Phase 0: Prepare

This phase deals with the planning that you need for the iteration. You must select the part of the application that you are going to modernize and define how you are going to measure your progress. Every iteration must have planning activities so you can effectively adapt to the immediate situation.

- ▶ Phase 1: Experimentation

You must try new things in each iteration. You must understand the technologies that are available in the market and determine whether they fit your situation. This phase is about experimenting.

- ▶ Phase 2: Off and running

This phase is about doing the real work and applying what you experiment with to achieve the goals of the iteration.

- ▶ Phase 3: Evaluate and expand, future thinking

This phase involves analyzing the work that is done so far, including all previous iterations. In this phase, you can see how to introduce new technologies to enable greater flexibility in the modernized application. This phase is dedicated to analyzing how to expand the current functions of the application.

- ▶ Phase 4: Deployment

You must deliver the software as soon as possible. Every cycle ends with working software. You must carefully select the deployment strategy to reduce risks.

2.3.3 General principles

This section provides some vital principles that you should follow during the modernization project. The application of these principles can help you with the modernization project, especially when you must face specific unexpected situations and make important decisions.

Iterative and incremental process

The modernization process can be thought of as an iterative and incremental process. It is iterative because the whole project can be separated into smaller projects that are easier to manage and faster to finish. It is incremental because with each finished iteration, you make the application more modern, one component at a time. This is also called an *agile* approach.

Sometimes, people that are involved in a modernization project try to follow a “big-bang” approach and end up being overwhelmed or frustrated. So, with each modernization project, remember to divide and conquer.

Keep researching and reading

Things change. What is cutting edge technology today can be obsolete in a matter of a few months. To keep yourself modern, make it a habit to read and research modernization theory and technologies. Read about the latest tendencies and try them. Go to seminars and conferences. Talk to your colleagues and look for successful experiences (or failures) about applying modernization approaches or technologies.

Modernization is not something that only you are dealing with. It affects many companies, platforms, and programming languages. Find time to experiment with technologies. Create proof of concepts to both learn and to prove the technology in your environment.

Automating what you can

Occasionally in a modernization project, you find yourself doing the same task over and over again. Think about how you can automate these steps. There are always better ways to do things. Look for commercial modernization tools or create them yourself.

Testing, testing, testing

Tests are the only way to ensure that you are not breaking anything that was not already broken. There are plenty of different testing approaches that you can use in your project. Here are two of the most common ones:

- ▶ Regression tests

This type of test focuses on comparing the results of an application against a pre-established base line, in this case, the original application. One key aspect of this type of test is that you do not need to understand the system before testing it. This can help when you are starting your modernization project and your comprehension of the code is not complete.

- ▶ Automatic unit tests

With unit tests, you can test a small part of the application. Unlike regression tests, unit tests require more understanding of the code that you are testing. So, start creating automatic unit tests as soon as possible. Unit tests must be designed in a self-configured way, meaning that you should not need to configure anything before running the tests. Also, the units must be able to run fast. You might need to run them after every compilation.

All of these approaches can complement each other, so figure out a way to use them in your modernization project and automate them. Remember, you should be able to run your tests fast and often; it is the only way to feel more confident while changing your code.

Do not overanalyze

Modernization is about experimentation. Sometimes fear can derail your efforts, so do not be afraid and do not overanalyze things. Start working as soon as you can.

The old code that you are modernizing causes surprises every occasionally, no matter how much you analyze it. Remember the iterative and incremental nature of the process and plan and analyze in the same way.

Being agile

Modernization projects must be agile, so focus on avoiding activities and documentation that do not add value to the process. Keep in mind the following agile principles:

- ▶ Deliver working software frequently.

Plan your iterations in a way that you can deliver working software fast and frequently. Your executives must believe in the value behind a modernization project, so do your best to deliver value to them in early stages of the process. A true agile project is broken down into two-week iterations. You are continually and regularly delivering progress. However, do not get hung up on breaking things into the smallest units possible. The purpose is to be flexible and deliver small incremental updates.

- ▶ Focus on the software over documentation.

Avoid any documentation that does not add value to the modernization process. Look for dynamic documentation that can be easily maintained and reused for different purposes. There are many content management systems that can help you create collaborative and dynamic documentation. Using a tool such as a wiki can be useful because it allows the user community to contribute to the documentation effort.

- ▶ Simplicity is essential

Keep things as simple as possible. Resist the temptation to add features that might be used in the future but are not needed immediately. Unnecessary complexity is a source of bugs that can damage your process.

- ▶ Give continuous attention to technical excellence.

Do not forget that modernization is about improving. Focus on applying better practices and technologies. Become an advocate of excellence. Promote preferred practices and transmit your knowledge. You must help others to understand the importance of doing things correctly.

- ▶ Business people and developers must work together.

Build good communication lines between developers and business experts. The application that you are modernizing is what supports the business. Do not underestimate the contributions that non-IT personnel can make to the project.

Always use modern tools

Stop using obsolete tools. Modern tools help you do things faster and better than before. Modern tools are valuable resources that can help accomplish amazing tasks. Do not let your current tools stop your modernization journey.

Naturally, you are more productive with tools with which you are familiar. Do not mistake this advantage of your familiarity as proof of the superiority of the tools. Changing tools require a period of adjustment before their productivity gains are realized. With time (and often it is not much time), modern tools allow you to be even more productive than before. It is vital to attract new talent to the organization, and new developers feel comfortable with tools that resemble the tools that they use. “Green screen” development tools can give the impression that you are using obsolete technology, so avoid using them. Many developers see significant productivity improvements, normally 20 - 50%, depending on each developer, by switching from a green-screen based development environment to a modern integrated development environment (IDE).

2.3.4 Phase 0: Prepare

Applications are like living beings. As they grow, processes, people, and other systems start to build around them. There is more than code in a modernization project and you must prepare accordingly.

The main objective of this phase is to prepare you for the modernization effort ahead of you. You need a clear, panoramic view of what the application means to the different stakeholders who are involved in the project.

Building the business case

Before you begin the project, you should have the application inventory prioritized. Now, you must create the business case. A business case can determine whether a project is cost-effective. The goal is to convert your needs into something that your manager can understand in their terms and make a business-based decision.

A business case contains the following parts:

- ▶ Problem statement

Ensure that this part describes the inadequacies, inefficiencies, and weaknesses of the current application. Put together numbers and figures that explain the situation to the decision makers and help them see the importance of the modernization project.

► Solution

This is the high-level description of the solution for the problem. You should include the following items:

- Current state-of-the-art technologies.
- An overview of the proposed solution.
- A preliminary cost and schedule proposal.

You can also include a brief description of the modernization model that is going to guide the project.

► Risks

You should do an honest assessment of the risk that you might have to face. This assessment helps you prepare yourself before any unexpected situations happen. Also, a good risk analysis helps you gain credibility by enumerating the things that might affect the scope of the project.

► Benefits

Ensure that you specify the benefits of the modernization project in a quantifiable way. It is recommended to include several scenarios:

- Pessimistic scenario: In this scenario, you should specify what are the minimum benefits of the modernization project. Ensure that you indicate what risks are more likely to happen and how you are planning to overcome these situations.
- Optimistic scenario: In this scenario, you can specify what are the maximum benefits of the modernization project. Be sure to set realistic quantifiable benefits. Avoid exaggerating, which can affect your credibility if something goes wrong.

► Metrics

You must define how you are going to measure your progress. Each iteration should be evaluated against these metrics. This is not only something that is important to your manager; it also helps you adjust your plan.

► Stakeholders

Identify all of the stakeholders of the project. Make sure that you include all the relevant people, from developers to business users. Having input from all the affected stakeholders ensures that you remember the importance of the application to your business.

► Examples

Here is where proof of concept comes into play. If you can show a small, successful example of what you are attempting to accomplish, it helps boost the business case.

The business case is essential for the modernization project. Without it, it is doubtful that you can obtain the funding that is required and keep it throughout the project's life. It is vital that the business case is complete. Otherwise, your modernization effort can be hampered even before you change a single line of code.

Building the team

Modernization projects always involve several people. It is vital that you define who is going to be part of your team at an early stage. Besides the developers that do the actual work, the modernization team is also composed of non-IT personnel (for example, the users). Do not underestimate their contributions or commitment to the project.

Make sure that you define the following roles at a minimum:

- ▶ **Modernization team lead**

This person is in charge of directing the team. Make sure that the person who is going to be assigned to this role is an advocate of modernization. It must be a pragmatic and open-minded person who is not afraid of change and passionate about continuous improvement.

- ▶ **Modernization engineer**

The modernization engineers are in charge of applying the modernization approach to the application. Make sure to choose the correct people for this role and keep them focused on the main objectives. During the modernization process, it is natural to want to fix everything immediately. Engineers must remember that modernization is an iterative process.

- ▶ **Business logic expert**

The business logic expert usually is a non-IT person. They can be final users or someone whose business process is supported by the application. This role is essential for the project. They can give you guidance about understanding the reasons behind specific functionalities of the application.

- ▶ **Application maintenance personal**

This is the person who has experience maintaining the application and modifying it according to the new requirements. This role can explain most of the design and coding decisions.

Do your best to maintain the team's cohesion and promote team work. Sometimes, the application maintenance personnel feel threatened by the modernization project. Make sure that you convince them of the importance and value of the project.

Developing the plan

Do not try to anticipate every little detail. Things most likely will change, so define a high-level roadmap of the project. Then, do a detailed plan for only one or two iterations. Use the roadmap as you perform your plan to keep your project on track.

This plan should define your immediate objectives for the iteration and the tasks that are needed to accomplish them. Make sure that your plan includes a deliverable at the end of the iteration.

Developing standards and processes

At first, you experiment frequently with the modernization process, but remember to document what does and does not work for you. During this process, you must define standards to ensure that the modernization team does things in a predictable way. As soon as possible, define a set of rules and standards that aim for quality. Be prepared to modify your standards as you and your team find improved techniques.

Always avoid reinventing the wheel

If there is a good standard in the industry, analyze it. If it fits your needs, adopt it. You are not the only one trying to modernize applications. Many things that you need already are researched and implemented by someone else.

2.3.5 Phase 1: Experimentation

This phase is a good point to try new technologies. Given that this is an experimentation phase, you perform many tests. Most of the things that you are going to code are not going to be in the final product.

The following sections describe some of the activities that you should consider in this phase.

Doing a proof of concept

You must know which technologies are best for your project. The preferred way to determine this goal is by creating a proof of concept (PoC). You might find obstacles and need to do some rollbacks, but the PoC helps you and your teammates gain experience.

If you are planning to select a commercial solution, it is likely that you are going to need your manager's approval. Do not rely solely on marketing figures and charts. Show them real working software. Take advantage of no-cost trials.

Remember to document what you learn in this phase. Make sure that you involve your stakeholder with the PoC. They can provide valuable insight to help make sure that you are going in the correct direction.

Looking for simpler projects

While you are learning and getting used to the modernization process, avoid selecting the most complex application. There should be applications that are important for the business and do not add unnecessary risks to the process.

With every success, gain more skills and confidence that enable you to modernize more critical and complex applications.

Experimenting

You are changing code, changing the UI, and changing the architecture. This is just testing and you are not going to deploy it to the production environment. The ideas and experiences that you gain in this activity help you work better on the new application and to understand the underlying code.

Reassessing

This whole experimentation process is going to broaden your thinking. Many ideas that you had are going to change. Do not be afraid to reassess your modernization approach if you think that there is a better way to do it.

Defining the measurement criteria

In this phase, you gain more knowledge about the application. Define the mechanisms that you are going to use to measure your progress. You are going to report regularly to your manager, and they need to see your progress. So, define your metrics according to the selected application and modernization approach.

2.3.6 Phase 2: Off and running

This phase focuses on doing real work. Now that you have a panoramic view of the available technologies and techniques, start modernizing the application.

Keep in mind the iterative and incremental nature of the modernization process, so start step-by-step in a way that makes sense for your situation. Divide the application into small units that you can modernize one at a time, but do not lose the whole view of the application.

The following sections describe some of the activities that you should consider in this phase.

Building your sandbox

Many modernization projects run in parallel with other business projects, so it is important that you build a sandbox that is specific for your project, where you can be sure that you are not affecting someone else's work and that no one else is affecting your work.

When you design and build your sandbox, remember to consider the following elements:

- ▶ Objects
- ▶ Data
- ▶ Tests

Objects

Copy all of the application objects to your sandbox, including, at the least, data areas, programs, service programs, user spaces, data queues, and any other object that your application is going to need to run in an isolated environment. To help you build the sandbox, create the application structure diagrams, which can be generated with automated tools.

Data

In addition to the executable objects, make sure to have all of the tables that interact with your application. Isolate the data to allow changes to be made without affecting other projects.

You should develop a method for restoring your sandbox data to its initial state in an automated fashion, which helps you run repeated test scenarios and automate testing. Always journal your sandbox data. Journals are indispensable when debugging your changes.

Tests

You need a complete set of tests to ensure that your modernization work is going in the planned direction. Your sandbox should contain a test set that you can run multiple times and easily. Without tests, any change can be unrecoverable.

In the first iterations, you must have a set of regression tests. As the project advances, your understanding of the application increases, enabling you to write specific unit tests.

Make sure that the test set covers all parts of the application that you are changing to ensure the “do no harm” approach.

Understanding the application

In every modernization project, you should take time to understand the application with which you are dealing. The level of understanding of the code that you need depends directly on the modernization approach that you select. Consider the following techniques for code understanding and apply them to your situation accordingly:

- ▶ **Compilation of existing documentation**

It is likely that there is no documentation of the original application. Do your best to compile every piece of documentation that is available. Some useful documentation might include the following items:

- Deployment and configuration notes
- Incident history
- Test cases

- ▶ **Code reading sessions**

The most basic technique to understand the code is to read the code. You must read the code that you are working on with your team, and look for repeated code and logic. Initially, reading the code does not appear to be helpful, but after you become familiar with it, things become clearer.

Ask the experts about the business functions that are implemented in the code and keep in mind those functions while reading the code. Sometimes it is easier to understand the underlying business process than the code that implements it.

- ▶ **Debugging sessions**

Debugging the application with specific tests can help you understand the application and how every module works. The usage of modern tools, such as IBM Rational Developer for i, makes the debugging sessions more productive.

- ▶ **Static analysis**

This is the analysis of the source code or the compiled object without running it. Usually, the static analysis has the following components:

- Usage of code analysis tools

There are several tool providers that have specialized tools to review and help you analyze your monolithic programs. The Rational ARCAD Power Pack provides the Observer tool, which can help analyze code.

- Extraction of business rules

Business rules refer to the business logic that is embedded in the code. Many tools help you extract this type of data from the code, which can help you understand the functionality that is implemented in the program.

- Dependency analysis between components

Refers to the identification of relationships between the modules of the application. The diagram that is generated can help you understand how one component integrates with the whole application.

- Internal structure diagram

Most legacy applications are large, monolithic programs. To easily understand the flow inside these programs, some tools can generate call diagrams between internal subroutines and subprocedures, such as the ARCAD Observer tool.

- Complexity and maintainability metrics

Complexity and maintainability metrics can help you diagnose the section of code on which you are working and measure your progress. They can also help you estimate the amount of work that is needed to modernize.

These types of analysis are most easily done through the usage of automated tools.

- ▶ **Dynamic analysis**

This type of analysis is performed by tracing the running of the programs in a specific environment. There are many uses of the results of the dynamic analysis:

- Code coverage

Traditional applications usually contain “dead code”. This is code that is not run but is still in the program. To identify code that is no longer needed, you can use a tool that helps you calculate how much code is covered with a set of transactions and how much code is possibly dead.

- Dynamic call diagrams

The preferred way to understand how the program works is by monitoring its behavior while running. Dynamic analysis tools usually provide data that depicts the execution flow of a program, subroutines, and subprocedures.

To be effective, the dynamic analysis requires a good set of tests to produce an interesting result.

Modernizing

When your sandbox is prepared and you have a better understanding of the application, you can start doing modernization work. Here are some of the common tasks that you can do in this stage:

- ▶ **Cleaning the code**

Cleaning code is the removal of elements that make the code hard to understand. Here are some of the things that must be considered when you clean your code:

- Removal of duplicated code.
- Conversion to a more readable form. In the context of RPG, this means converting it to free format.
- Removal of unused variables and procedures.

- ▶ **Restructuring the code**

After you clean the code, you can extract related business logic of service programs that can be reused in different parts of the application. There are several good tools that are available to help you identify and restructure your code. Here are some tasks that you can do in this step:

- Extraction of similar logic into reusable components.
- Encapsulation of global variables.
- Conversion of subroutines to procedures.
- Renaming of variables and subprocedures to more meaningful names.
- Writing self-documenting code.

- ▶ **Repackaging**

Every module must contain only functionally related subprocedures and every service program must contain only functionally related modules. You must separate monolithic modules and organize the procedures into new modules. Every module must have one responsibility.

- ▶ **Modernizing the database**

Data base modernization is a complete modernization model by itself. Later in this book (Chapter 9, “Database re-engineering” on page 351), you find a methodology that guides you through the process for this modernization.

Testing

After you modernize the application, make sure that all of your tests are successful. If your tests are not successful, you should not continue with the modernization process. You must ensure that everything is working as before and that you are not breaking existing functions.

2.3.7 Phase 3: Evaluate and expand, future thinking

The main objective of this phase is to maximize some of the benefits of the current state of the art technologies that are applied to your application.

You have prepared your environment and you have modernized the application. Now, it is time to evaluate the progress of your work. If this evaluation indicates that you should step back in the process, do not worry; you might have to do this step multiple times.

If the evaluation shows that you have accomplished your goals, it is time to start expanding the value of the application and prepare it for the future.

The following sections describe some of the activities that you should consider in this phase.

Measuring

You must determine whether you are achieving the modernization objectives. If your main goal is to reduce complexity, this step must reflect a reduction of the complexity. If you must improve the user experience, you should measure it in some way. The key point is that you keep control of the objectives and the progress to achieve these goals.

Your measurement should indicate adjustments that should be made to the project.

Improving the architecture

Now, you can consider optimizing the application architecture. You might want to start separating the application in layers or tiers and to include more components, such as a new UI replacement layer. This step focuses on the application as a whole.

Integrating

One of the common modernization goals is to integrate the application in to an enterprise architecture. Often, it is necessary to enable the application for SOA or to make it more flexible for usage with other enterprise integration technologies.

Bringing in new technologies

If there is any new technology that can help you improve the application, you can include it in this step. Carefully analyze it to keep it from becoming a future obstacle.

Always aim for standard tools, technologies, and well-tested solutions. Evaluate the user community around them and the support of the official application maintenance personnel.

These new technologies can become obsolete quickly, so wrap them in a way that helps you replace them easily if necessary.

2.3.8 Phase 4: Deployment

Now, it is time to deliver the results of your work. However, the components that you deliver should not be considered a final product. With each iteration, you can improve pieces of software even if you already delivered them. The key point here is that you do your best to start deploying as soon as you can to show the value of your modernization effort.

The following sections describe some of the activities that you should consider in this phase.

Documenting the application

It is likely that the new application is different from the old one. You must create effective documentation that explains the new application to the people that do maintenance and the users and stakeholders of your applications.

Take advantage of any available resource that can help you document. Use class, sequence, and deployment diagrams to help people understand the application. Most types of Unified Modeling Language (UML) diagrams are designed for object-oriented languages, but you can adapt them easily to your situation. The advantage of using a standard type of diagram is that many people can understand them easily, which makes you seem more professional.

Your documentation must add value. So, avoid documentation that goes directly to the shelf. The usage of content management systems (for example, MediaWiki, WordPress, and Joomla) can help you create simple and collaborative documentation.

The documentation must be easily maintained. There are many tools that can generate documentation from the code or from the compiled objects. Use them (or create them) if possible.

Training the people

Depending on the modernization approach, you might have to train the users, the application maintenance personnel, or both.

The documentation that you previously built should be used as the primary reference for the trainees. Make sure that this documentation can be accessed easily and that it contains all the necessary information for the users and for the IT experts.

If the new application is more difficult to understand than the original, this is a good indication that the modernization plan needs improvement. Evaluate the new system from different points of view.

Defining your deployment strategy

Just as there are many modernization approaches, there is more than one way to deploy the new application. Make sure that both approaches are aligned. Here are some of the most common deployment techniques:

- ▶ Big-bang approach

This approach for deployment replaces the entire application at one time. This approach is used with modernization projects that must solve an immediate problem affecting critical services. However, the associated risk is higher and must be carefully managed.

When you follow this type of approach, it is necessary to maintain the old application and the new application in parallel. The issue is that changes in the new system should be reflected in the old one and vice versa. This situation can lead to much work and effort.

► **Incremental approach**

Also known as “Phased-out”, in this approach, sections of the application are re-engineered and added incrementally, which allows you to deliver results and identify errors more quickly.

When you follow this approach, you can change only one component at a time without considering the application as a whole. Thus, the modernization process might take longer to finish.

► **Evolutionary approach**

Similar to the incremental approach, the evolutionary approach replaces sections of the original application with modernized parts. The difference is that in this approach, the sections to replace are selected based on their functions and not the structure of the application. This situation means that the developers put forth extra effort to identify functional parts that are separated in the original application.

To help you understand the differences in these approaches, consider Table 2-1, which contains a brief comparison of the three approaches.

Table 2-1 Deployment approaches comparison

Component	Big-bang	Incremental	Evolutionary
Replacement	All at once	Structure	Functionality
Overhead	Low	High	High
Risk	High	Low	Medium
Time to deploy	Fast	Slow	Slow
Time to modernize	Slow	Fast	Fast

2.3.9 Repeat as necessary

After you finish the process, you must go back and repeat the process as many times as needed. In every iteration, a new component is modernized and you are closer to your goals.

Not all the phases require the same amount of effort in every iteration. At the beginning, you probably must invest more effort in the preparation and experimentation phases. However, as the project advances, these phases are shorter. You see how the phases adapt during the successive iterations.

Evaluate your progress in every iteration and adjust your path accordingly. While going through all the phases, adhere to the principles of modernization and adjust the process to your situation.



Modern application architecture techniques

This chapter introduces application architecture techniques that can help you design and build modern applications.

3.1 Introduction

What is a modern application? How can you define a modern design? Given the pace of changes in technology, the term *modern* can be hard to define. Every day, new technologies are built, patterns are designed, and recommended practices are refined or changed. Given the dynamic nature of software development, it is easier to understand what is a bad design.

A bad design exhibits many characteristics that you want to avoid in the software that you build. In this chapter, you learn some of the characteristics of what contributes to bad design and some design techniques that you can use to avoid them.

3.1.1 Symptoms of a bad design

On the first release of an application, programmers have a clear view of what the software is and how it is built. If they need to make a change, it is often easy for them because of their familiarity with the code. As time goes by and multiple programmers start making changes, new additions, and updates, the application starts to deteriorate. Why does this happen? Why does software degrade over time? There can be many reasons and sometimes this degradation can be difficult to avoid, but you should focus on creating a good design that is harder to break over time.

If your application exhibits one of the following symptoms, you are probably facing bad design:

- ▶ Rigidity

The application is hard to change because a single change can trigger a cascade of multiple changes. With rigid designs, what appears to be a simple change can convert to a complex project that is difficult to estimate. The more modules that are affected by a change, the more rigid the design.

- ▶ Fragility

Changes cause the system to break in many places when a single change is made. These parts have no conceptual relationship with the area that was changed. Fixing one problem can cause new problems.

- ▶ Immobility

A design is immobile when it contains parts that might be reusable, but the effort to separate them from the original system are too great. The application is too tangled to reuse parts of it.

- ▶ Viscosity

When you are doing a change in an application, you have two options: preserving the design or “hack” and break the design. Viscosity makes it harder to follow the design preserving method. Over time, developers tend to start doing “hacks” more frequently to the application because of a lack of understanding and time pressures.

- ▶ Needless complexity

Software design is essentially a creative process. But sometimes, a design contains elements that are not currently useful. Anticipating requirements might seem like a good thing, but if this code is not carefully planned, it converts into source with complexities that do not add benefit.

- ▶ Needless repetition

The design contains repeated structures that can be unified under a single reusable component. Sometimes programmers use a “copy-paste” approach for coding, which can lead to redundant code that makes it difficult to do changes and correct bugs.

- ▶ **Opacity**

The design and the code are hard to read and understand. It does not express its purpose correctly.

In the following sections, you learn techniques that can help you to create modern designs that attempt to avoid these problems. When you read these sections, keep in mind the bad design symptoms that are described here and discover how these preferred practices can help you avoid them.

3.2 Modularization

Have you ever had to deal with a large, monolithic program? Many traditional legacy applications are monolithic programs that are composed of one single module. In this single large module, you can often find code to handle the display, user interactions, business logic, and database access. There is no sense of separation.

Maintaining monolithic programs is no easy task. Given their size and complexity, programmers face many obstacles. Often, they need the help of the original author of the program to understand it (assuming the original author is still around). As a result, maintenance projects and new additions tend to grow in time, cost, complexity, and risk.

What do you do with these applications? So far in this book, you have been learning about modernizing your applications. Monolithic programs must be modernized but in parallel with an overall modernization project. You must ensure that the new applications are not designed with this approach. Modularization is the key to designing easier-to-maintain applications.

Modularity can be defined as a quantity of components within an application that are separated and recombined. Modularization is a software technique that focuses on separating the functions of a program into smaller reusable modules containing only one function that is then used throughout the whole application.

To achieve a good modular design, you must start thinking about separating functions and implementing just one function per module. If a module does many unrelated functions, you should reconsider your design.

3.2.1 Why modularization is important

Modularization brings many benefits to the application and its programmers. In this section, you learn some of the main benefits that modularization provides.

Easier to change

You probably know by now that applications change. A business is a dynamic entity and so are the applications that support it. Modularization reduces the difficulty, cost, and time to do changes. With monolithic applications, even a simple change can become an arduous task requiring more effort and risk than expected.

How does modularization make changes easier to do? Imagine a huge monolithic application where all the functions are in the same module. The business needs a small change in the logic that is embedded in the code. You are not familiar with the whole application and time is against you. This scenario is not far from reality. When programmers face this situation, they do not have the time to understand all the mixed code and make the change using bad techniques such as “copy-paste”. The risk increases and the code becomes a bigger mess than before.

Now consider the other scenario. You must do the same small change, but in a well-designed modular application. If you generate a diagram of the application, you see how the application is structured. Now, you can understand which modules you must change. You did not have to go through the thousand of lines of code that were not related to your change. Modularization makes it easier to understand the whole application and focus only on the parts that you care about.

Now, consider an example where you must make a change in how you access a database. With a monolithic application, you might have to repeat the same change many times. What if you miss one of the data access locations? You just injected a bug. With a modular approach, you update the one module that deals with the database access and you are done.

Easier to understand

When you are trying to understand a monolithic program, you must see everything at once. The smallest component is the whole application. It is not easy to focus on the UI logic or the database access separately. Given the absence of layers and separation, you are forced to understand all aspects of the application even if you are only interested in a specific part.

In a modular application, you can start learning different parts of the code and ignoring the rest. For example, you can focus on understanding how a specific part of the business logic works, without having to go into the UI or data access.

This ease in understanding benefits you mainly in the following areas:

- ▶ **New talent training**
Less time and costs to train new people in the application. It is easier to explain small pieces one at a time than a whole monolithic application.
- ▶ **Modernization projects**
These projects require personnel who understand the original application before it is modernized. Code understanding is an arduous task that can be eased with a modular design.
- ▶ **Error location and fixing**
Finding errors in monolithic applications can be difficult. A modular design can help you isolate the errors easier and faster.

Easier to test

From a testing perspective, it is easier to test modular applications. Similar to the understanding process, you can focus on a particular module and write unit tests for it. Unit tests are valuable resources to validate your code. They allow you to write tests for specific sections of code that can be hard to reach. Unit tests must be complemented with integration and function tests, but in monolithic applications you do not have this option.

In terms of test design, monolithic applications tend to have higher complexity, which makes them harder to test. This difficulty in testing can lead to important sections not being tested correctly and additional risk for bugs in the future.

Reusability

Large pieces of software are harder to reuse. Modularization aims to create the application as a composition of small pieces. With a good design, it is easier to reuse these pieces in different parts of the application.

3.2.2 General principles

You have learned what modularization is and why it is important, but now you learn the most important principles that you need to keep in mind when defining how to modularize. Splitting the application into smaller pieces is essential in the path to modernization, but sometimes the difficult part is knowing what to include in each piece. This section addresses this point. Keep in mind the guidelines that are described in this section when you design a modular application.

Single responsibility

This principle states that every module of an application must have one responsibility. A responsibility can be understood as a reason to change. For example, consider a module that retrieves data from a table and also generates a report that is based on the data. If a change that is related to the data retrieval must be made, the entire module must change. But if a change must be done to the report, the same module must change. Therefore, the module has two unrelated reasons to change. The correct way is to create two modules, one focused on retrieving data and other focused on generating the report.

When a module has more than one responsibility, it is easier to cause collateral damage: You are changing the report, but the data retrieval function is also affected.

High cohesion

Cohesion is the degree in which elements of a module belong together. It is the functional relatedness of the elements within a module. You can detect low cohesion when a module has more than one responsibility. Also, low cohesion can be seen when common responsibilities are separated in different, unrelated modules.

You should design components with cohesion, which makes them more robust and easier to reuse. All that you have to do is put related functions in the same module and make sure that future changes take this function into consideration.

Loose coupling

Coupling can be defined as the degree to which each module relies on other modules. It is a measure of the strength of association that is established by a connection from one module to another. Tight coupling makes a program harder to understand, change, or correct by itself. A change in one module generates many changes in unrelated modules. It is like domino pieces falling in sequence.

Achieving a design with no coupling is hard and not practical, but you must try to focus on designing with low coupling in mind. Loose coupling can be achieved by creating high cohesion modules that communicate with others through a defined interface, hiding its internal implementation. Modules must hide their global variables, data format, and any other implementation details from other modules. Modules are called and the function within the module is affected even though only data passed in the defined interface.

Packing together only related modules

When you package your modules (service programs in the ILE context), remember to pack together only related modules. This action can be thought as the principle of high cohesion, but at the package level. Why is it important to design packages with this approach? If you pack unrelated modules together, you generate many dependencies to the package from different applications. A change of a module requires modifying the whole package, increasing the risk of damaging other applications.

Correct naming convention

Naming conventions must be defined in a way that helps the programmer understand the modular approach. You should define a self-documented naming convention that helps quickly identify the single function of a module. This convention helps others to maintain the order and style of the whole application. Good naming conventions also make tools, such as UML diagrams, more useful.

3.3 Tiered design

Two of the main objectives of modern application architecture are to maximize code reusability and to minimize the impact of changes and maintenance.

Over the years, there have been a number of different architectures, such as client/server, three-tier, n-tier, multitiered, model-view-controller (MVC), and rich client to service-oriented (SOA).

Regardless of which of these architectures you select, they all share one common feature: They attempt to separate the application into distinct tiers.

For example, a three-tier design separates the presentation, logic, and data layers:

- ▶ The *presentation* tier interfaces with the user. This might be a 5250-style interface, a web interface, or a mobile interface.
- ▶ The *logic* tier coordinates the moving and processing of data between the presentation and data layers. Processes consist of any decisions and calculations that must be made. This is where all the business logic must exist.
- ▶ The *data* tier is where data is stored and retrieved from a database. The data tier ensures the integrity of the data without any dependency on the logic tier.

Each tier is an entity unto itself and has no knowledge of the contents or methodologies of the other tiers. There are published interfaces that allow for communication between tiers.

Theoretically, any tier can be changed or replaced without affecting any of the other tiers in the application.

Assuming that the current presentation layer is a traditional 5250-style interface, the concept of a tiered design means that a new presentation layer (for example, a web interface or a mobile interface) can be introduced without having to make any changes to the logic or data layers.

Although this example is at an application level, the principle can be carried down into each of the tiers, which can themselves be tiered. This principle can be carried down to the programming level, where encapsulation is used to implement a tiered design within the code.

The implementation of a tiered design ensures that changing or adding a tier (at the application or program level) has no affect on the other tiers. Therefore, the main objectives of maximizing code reusability and minimizing the impact of changes and maintenance are met.

The exact architecture to be used can sometimes be influenced by other factors, such as the ISV tools, programming languages, and frameworks that are being used. Whichever architecture is being used, they all aim to achieve the same tiered principles.

A number of articles in iProdeveloper demonstrate the implementation of a tiered design using an MVC approach:

- ▶ In the article “Pattern Recognition: Ease Modern RPG Programming”, Scott Klement demonstrated an MVC methodology that is used with traditional 5250-style customer maintenance programs. These programs separated all database handling into a module in a service program, with the service program being a data tier. You can read the article at the following link:

<http://iprodeveloper.com/rpg-programming/pattern-recognition-ease-modern-rpg-programming>

- ▶ In the following article, “Pattern Recognition: Adopting The Pattern”, Paul Tuohy demonstrated how the same data tier was used with a web interface written in CGIDEV2. You can read the article at the following link:

<http://iprodeveloper.com/rpg-programming/pattern-recognition-adopting-pattern>

- ▶ Finally, in the article “Pattern Recognition: The Maintenance Benefit”, Paul Tuohy showed how the data tier can be rewritten without affecting either of the two presentation tiers (5250 or web). You can read the article at the following link:

<http://iprodeveloper.com/application-development/pattern-recognition-maintenance-benefit>

A tiered approach, at all levels within an application, provides the most flexibility in terms of maintenance and the potential for rapid change.

3.4 Business object model

When you develop your software architecture, think of the problem as a set of business objects and their interactions. A business object is often defined as an “actor” or “participant” in the business logic for your organization. It can be any type of entity that can be seen in the processes that drive the business. Common business objects include things such as employees, customers, work orders, receipts, and account records.

These business objects can be modeled as a set of rules, work flows, and data flows. Consider an example scenario where a customer calls to report a problem. The customer service representative might create a work order for a specialist to investigate further. A call log might be created and stored in a local database. The customer is a business object, as are the customer service representative, the work order, and the call log. Rules might also govern these transactions. For example, customer attributes might dictate the priority of the work order. Data also must be fetched, created, and modified (also part of the model).

As you can imagine, a business object model is done in terms non-programmers can understand. It is the business perspective of the problem or solution. As such, the creation and revision of the model can involve any set of technical or non-technical people from your organization.

3.4.1 Implementation

After a business model is created, it can be used to help you architect your software solution. As you can imagine, this type of model encourages object-oriented design. But how does it translate to your software solution? Here is a strict approach.

Each type of business object (BO) can be represented by an instance of a class. This class can (and should) encapsulate all of the attributes of that particular BO. Because each one has attributes that can be retrieved or changed as needed, these classes are often implemented as “data beans” with a simple set of entry points for creating or modifying attributes. The business logic is then written as interactions between these business objects. For interaction with data (such as a DBMS instance), the model includes data access objects (DAOs), which provide the interface to the data back end. The DAO provides a layer of abstraction so that the business objects do not need to understand the complexity of the back end. That advanced knowledge (for example, the specific format of a database's relational tables) can be encapsulated in the DAO. Data flowing between parts of a process can be further encapsulated into a data transfer object (DTO), which contains the data of interest (customer address) and nothing else.

Now, let us revisit our earlier example. When a customer calls in to report a problem, the customer service representative might need to look up the customer's address. The representative (being an “actor” in the process and therefore a business object) asks the DAO for this information. The DAO understands the complexity of the back end, fetches the data, and returns a DTO that contains the customer's address.

Of course, the real-life implementation of code that is based on the business object model is rarely as perfect as the preceding description asserts. For example, it might not be feasible to fully encapsulate a business object into a single class. Performance considerations or third-party software might impose restrictions on your design. Regardless, the model can still be used to help partition the design into relevant business objects and maintain a logical and meaningful separation of work.

3.4.2 Advantages

As you can see, the usage of a business object model helps the solution follow many of the preferred practices that are outlined in this chapter. For example, the model tends to enforce a strong notion of modularization because each BO, DTO, and DAO contain the information that it needs to complete its own necessary tasks. Further, each object does not need to contain detailed knowledge about tasks that are performed by other objects (single responsibility). In a good design, parts of the model can be replaced or modified without impacting the rest of the solution (loose coupling), which results in code that is more maintainable, reusable, and flexible.

In addition, modeling with business objects allows you to quickly identify dependencies when making a change. In this example, if the DTO retrieves a customer's address changes, what other objects might be affected? Similarly, it can help identify performance-sensitive modules within the solution.

As mentioned earlier, the business object model itself is not programmer speak. It is not class diagrams, function prototypes, or XML. In fact, it is often done with stick-men and circles. As such, just about anyone can contribute to the model by identifying interactions, associations, and rules for the model.

Most importantly, this approach forces the software architect to keep a business-level perspective on the problem. After all, when designing a software component, it is essential to understand how it fits into the big picture. This understanding is sure to result in a better software solution.

3.5 Data-centric programming

Before you learn about the concept of data-centric programming, you should understand the traditional approach for developing applications on IBM i: program-centric programming.

With this approach, the program has the following responsibilities:

- ▶ **Data Access**

Determine the files that are needed and the data access method (usually, record access level (RLA) to retrieve data one row at a time). Also, the program itself determines the relationship between the files, making the database management system (DBMS) of little help.

- ▶ **Business Logic**

The program has all the business rules embedded into the code (either directly or through a copy member). Business logic enforces the referential integrity between files.

- ▶ **User Interface**

Show the results of the processing of the data and business rules to the user.

Monolithic applications can be considered as program-centric, but you can also think of modular applications that are built as program-centric. Figure 3-1 shows an example of the structure of an application that is designed using a program-centric approach.

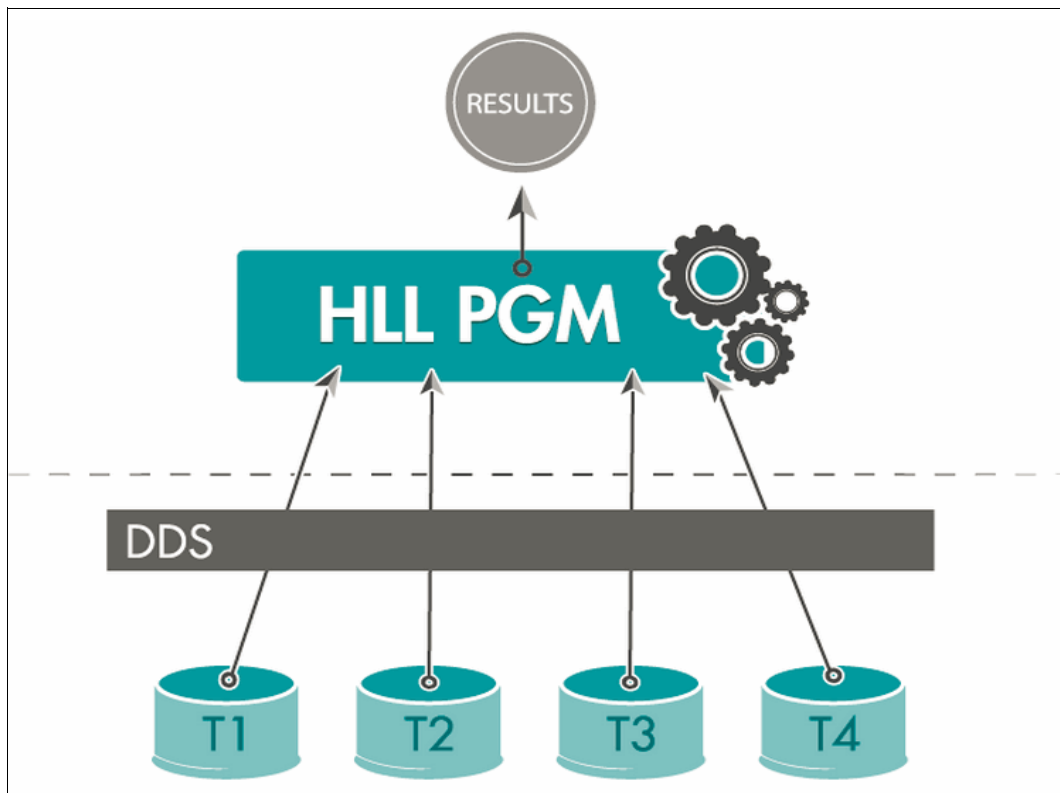


Figure 3-1 Program-centric design

As you might notice, the program-centric approach has many disadvantages:

- ▶ The application is not taking advantage of the DBMS and its future updates and features.
- ▶ It is inflexible and cannot adapt to new business needs.

- ▶ Business rules are duplicated across various programs. As business rules change, programs must be changed or recompiled.
- ▶ There are performance issues that are associated with RLA.
- ▶ The application must be recompiled if a file changes.
- ▶ Database normalization can be nonexistent.
- ▶ Referential integrity is implicit.

Conversely, data-centric programming can be understood as the strategy to solve business problems by taking advantage of the features of the DBMS. This approach proposes implementing more business logic in the database and separating the business logic from high-level application programs. Figure 3-2 shows the basic structure of a data-centric program.

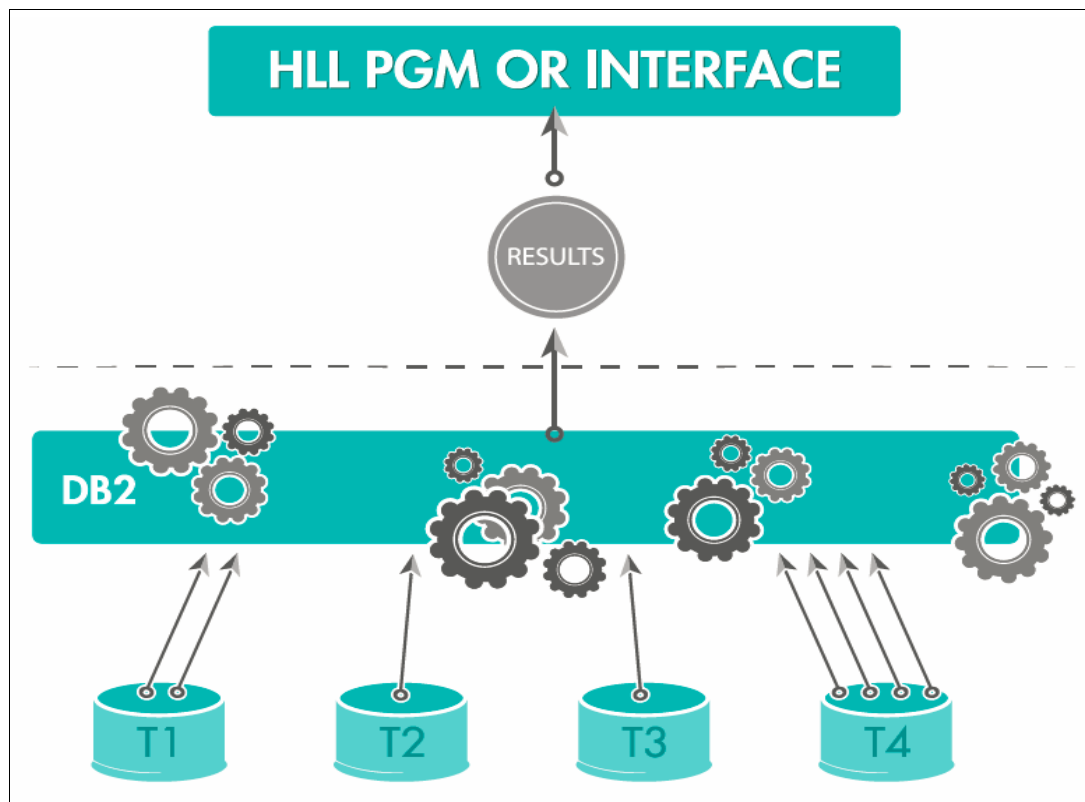


Figure 3-2 Data-centric programming

Data-centric programming aims to take advantage of the DBMS, specifically with the following items:

- ▶ Optimizing the access to the data
The application requests data through SQL statements. Then, the DBMS selects the best way to access the data and advises the usage of any indexes that are required to make it more efficient.
- ▶ Using constraints
With the usage of constraints, you can move business rules from the applications to the DBMS.

► Normalization

With the usage of the DBMS, it is easier to keep normalization at least to third normal form. The DBMS understands the relationships of the tables by use of primary key (PK) and foreign key (FK) constraints. PK/FK relationships are non-business data, usually through the usage of identity columns in each table. An identity column automatically increments when new rows are added to the table, and it is maintained by the DBMS, not by the programmer.

In summary, here are the primary goals of the data-centric approach:

- Drive as much work down into the database management system as possible.
- Define business rules as part of the database.
- The rules apply to all application interfaces.
- Take advantage of SQL-only capabilities.
- DDL modifications without affecting programs (that is, Index Advisor and others)
- Evolve the database to meet new requirements.
- Take advantage of new technology and enhancements to DBMS, such as the new optimizer (SQE).

The closer that your design meets the above data-centric goals, the more you can take advantage of advanced technology. In Figure 3-3, you see a comparison between program-centric and data-centric approaches.

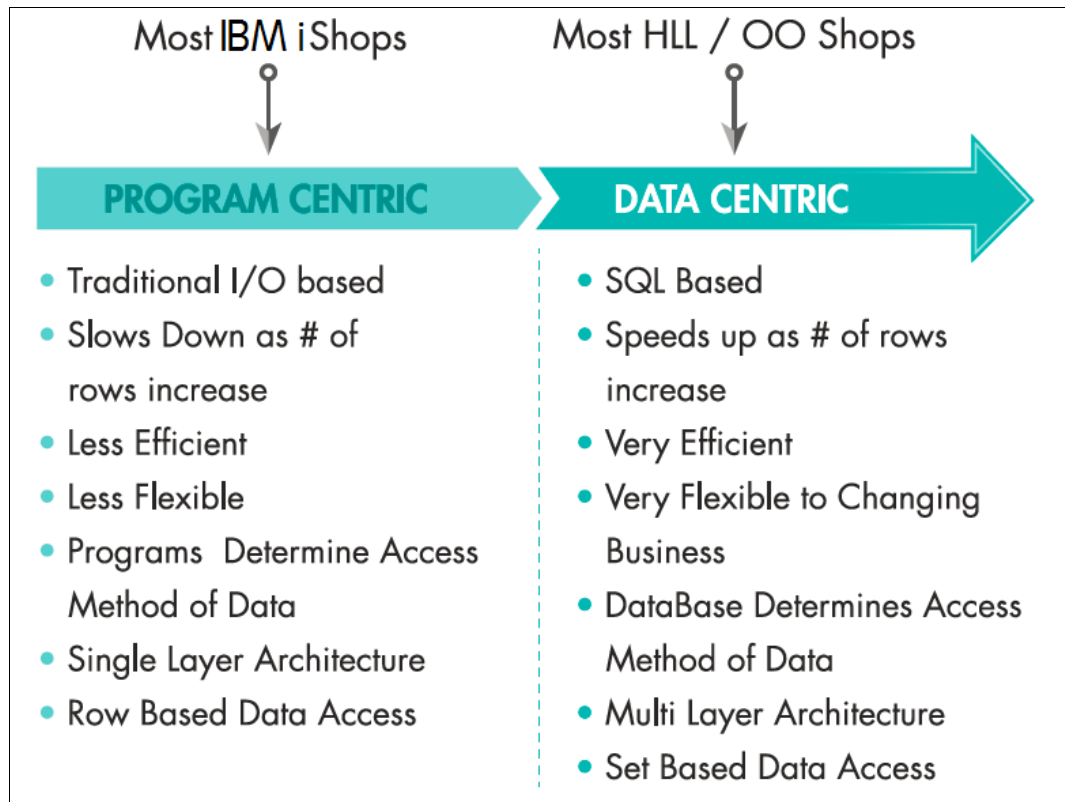


Figure 3-3 Program-centric and data-centric comparison

3.5.1 Moving to data-centric programming

If you are planning to move to a data-centric approach, there are some things that you must consider that can help you along the way:

- ▶ Referential integrity
- ▶ Check constraints
- ▶ Column-level security
- ▶ Column encryption
- ▶ Automatic key generation
- ▶ Triggers
- ▶ Stored procedures

Referential integrity from the start

It is important to start defining the relationships of the tables to ensure database integrity. For example, when you want to add an employee to a database and you must validate the employee's department, you can define a rule of referential integrity, which does not permit the addition of the employee record if the department ID is not correct or does not exist in the department table. With this rule, you ensure that there are no employees with the wrong department ID; this rule applies for all interfaces.

Triggers are an outstanding complement

There is business logic that can be implemented with triggers. Defining a trigger is a technique to ensure important business validations.

You might need to include a few validations that are not possible to ensure referential integrity. This situation is when you use triggers, which are the greatest complement to run the validations from the database.

For example, imagine that you want to validate the credit status of one client before accepting an order. This validation is not possible from referential integrity, but you can create a trigger to insert a delay to determine whether to accept the client order.

Figure 3-4 shows another example. When a new order is inserted, a trigger starts and the trigger receives information about the order and the customer. Immediately, an email is automatically sent.

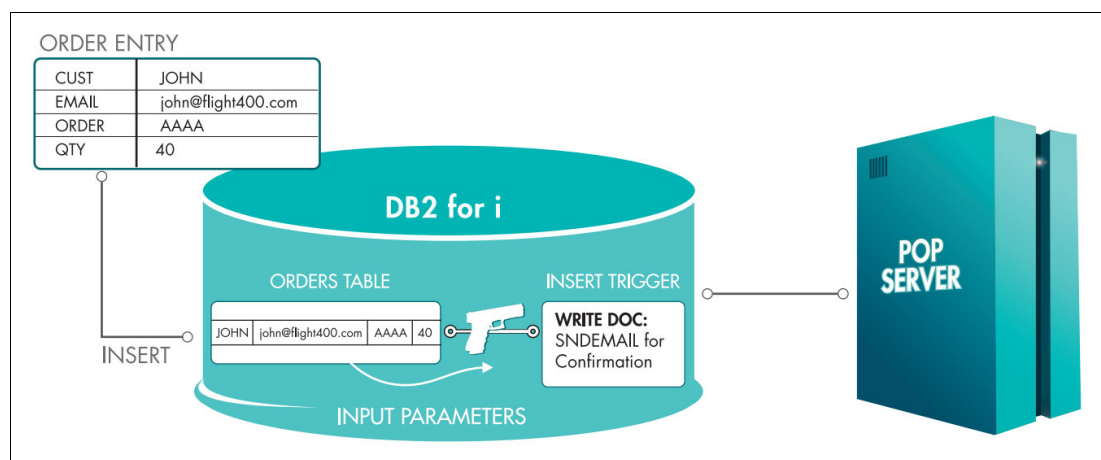


Figure 3-4 Trigger - conceptual example

SQL is the basis

SQL is the basis of data-centric programming. In this programming, SQL works efficiently at processing large sets of data.

All objects of the database are most effective when they are created with SQL.

You should consider creating all objects with SQL because this can improve the data access and performance of the applications.

3.6 Service orientation

Figure 3-5 shows the evolution of applications. At the beginning, applications had all the code for all functions in the same place. The maintenance for these applications was difficult because of the quantity of code and the functions in the programs.

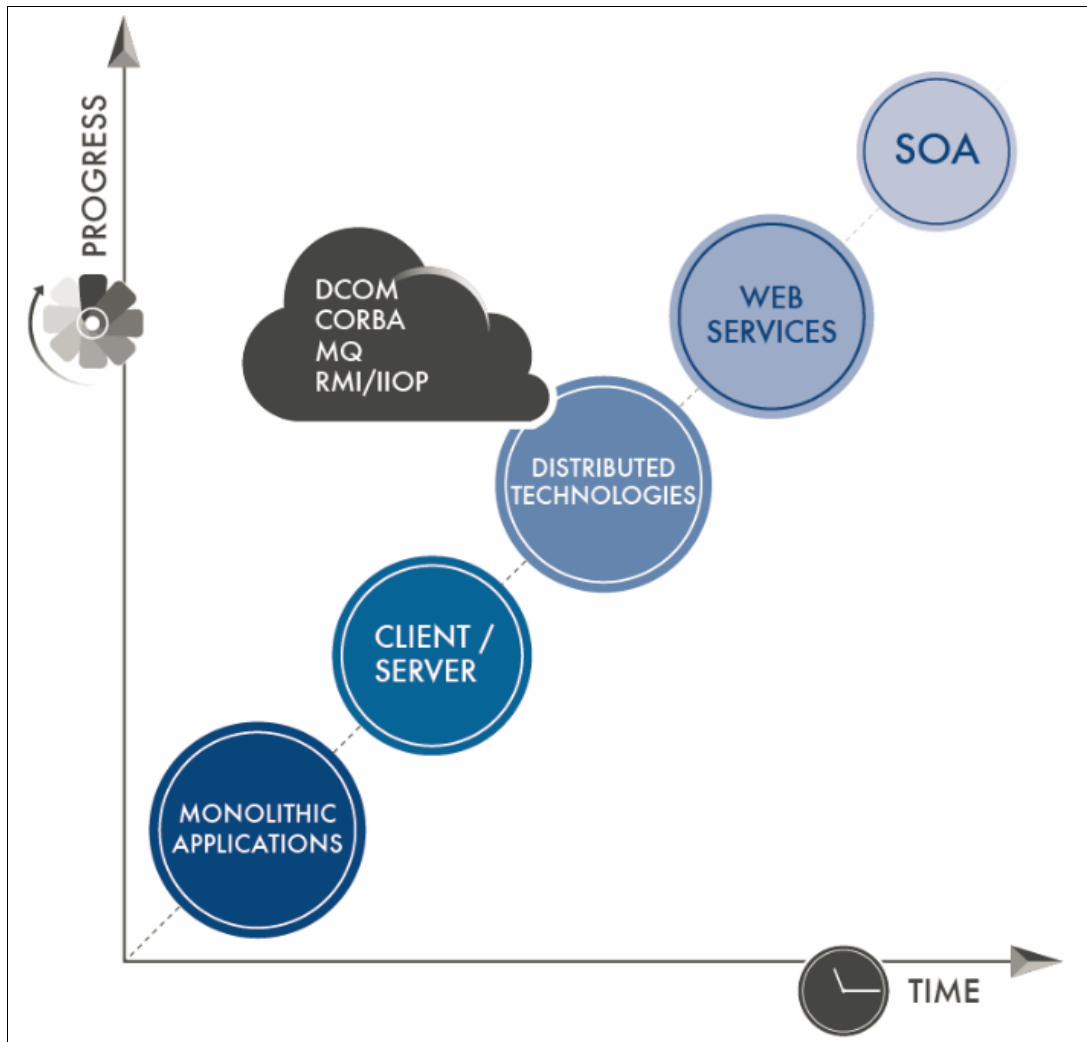


Figure 3-5 The history of programming methodology

After that, distributed applications were considered. These applications have one part of the code on the server side and the other part on the client side. This period was called the time of the client/server applications. The client/server applications were useful for user interfaces, but they required a good design and good capacity or the application easily could get out of sync and cause issues.

After the client/server applications, the programmers developed applications by using different technologies. Some of these technologies are RMI/IIOP, CORBA, DCOM, and others. This period was called the time of the distributed technologies. During this time, applications were running over heterogeneous platforms.

Developing applications with technologies such as RMI/IIOP, CORBA, DCOM, and other similar technologies, is a difficult task. You should have high-level skills in using these technologies. For this reason, applications started to be implemented with the usage of web services. When you create applications that are based on service-oriented architecture (SOA), you are using a standard language called XML. XML is used to define the input and output of the services call, which can help decouple and promote a tiered approach because the caller of the web service does not need to know anything about the platform.

When we talk about service orientation in this book, we refer to SOA. SOA is an integration architecture approach that is based on the concept of a service. The business and infrastructure functions that are required to build distributed systems are provided as services that collectively, or individually, deliver application functions to user applications or other services.

SOA specifies that within any given architecture that there should be a consistent mechanism for services to communicate. That mechanism should be loosely coupled and support the usage of explicit interfaces.

SOA brings the benefits of loose coupling and encapsulation to integration at an enterprise level. It applies successful concepts that are provided by object-oriented development, Component Based Design, and Enterprise Application Integration technology, to an architectural approach for IT system integration.

Services are the building blocks of SOA, providing interfaces to functions out of which distributed systems can be built. Services can be started independently by either external or internal service consumers to process simple functions, or can be chained together to form more complex functions and to quickly devise new functions.

3.6.1 What is a service

A service can be defined as any discrete function that can be offered to an external or internal consumer. This service can be an individual business function or a collection of functions that together form a process. It can be compared to a module that is packaged into a service program. The difference is that a service is callable from anywhere.

The services are functions or operations that are available across a network. Some examples are Login Service, Calculate TRM, Impression Service, and so on.

The services are accessed independently of implementation or transport and are integrated across the network or enterprise, as shown in Figure 3-6 on page 43.

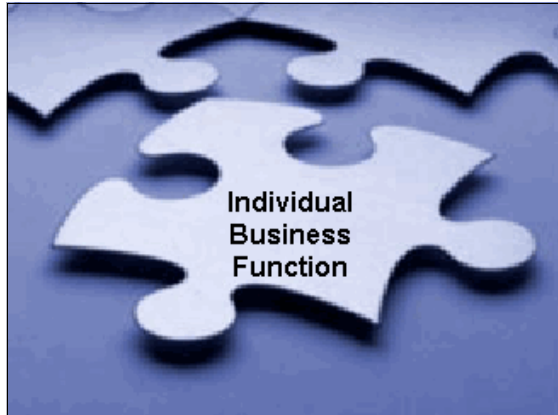


Figure 3-6 Service is like a piece of a puzzle

3.6.2 The properties of the services

A service must have with the characteristics that are defined in this section.

Loosely coupled

Services are defined by explicit, implementation-independent interfaces. Consumers of the service only depend on such an interface for service invocation; they are not concerned with the service implementation or even where it runs. Loose coupling promotes flexibility in changing the service implementation without impacting the service consumers.

Reusable

It is important to design services with reuse in mind and anticipate the different reuse scenarios. That is, each service has been designed for reuse, not duplication.

To design for reuse, complete the following steps:

1. Decompose the business need into its necessary function and services (separation of concerns).
2. Reuse and create business application-specific functions and services (create and reuse services).
3. Use common services that are provided by the operating environment (use the infrastructure).

The result is a composite application at run time representing a business process.

Encapsulated

Services should not physically expose any implementation details or deployment details within their interface design.

The concept of encapsulation by using interfaces should be familiar from the disciplines of Object-Oriented Analysis and Design (OOAD). Interfaces are used to separate publicly available behavior from the implementation of that behavior. Consumers of that behavior depend only on the interface that describes the behavior; the consumers are then not impacted if the implementation of that behavior changes in any way. A service-based architecture provide a set of coarse-grained interfaces that encapsulate access to finer-grained components and are accessible over the network.

Stateless

The service implementations should not hold a conversational state across multiple requests. Services should communicate complete information at each request and each operation should be functionally isolated (separate and independent).

High cohesion

The services interfaces should be concise, related, and complete sets of operations. Concise and complete implies every operation that is needed and no more.

3.7 Cloud

The fact that the IBM i operating system (OS) is one of the best multi-workload, multiuser business platforms is undeniable. The value of IBM i and its architecture is that it is based on object security, job descriptions, and robust job scheduling and work management. These features make it an ideal platform for running multiprocess workloads. IBM i users run batch processing, web, and interactive application workloads concurrently, while still taking advantage of high availability. Other platforms are not structured in such a way to run these types of workloads as efficiently and with job isolation (separation).

From a technical perspective, security and isolation of tenants from each other's business processes and data is a primary concern for the cloud. New tenant provisioning is the ability to create the infrastructure, platform, and software environment in a timely and repeatable manner for your new customers.

From a commercial perspective, the metering and measurement for billing is the key concern.

Here are the main points that you must consider for the cloud:

- ▶ Virtualization
- ▶ Multitenancy
- ▶ Performance
- ▶ Security
- ▶ Measurement for your licensing model

IBM i has a long history of virtualization. For example, Logical Partitioning (LPAR) capabilities have been available since 2001.

For multitenancy, when one user or 1000 users use the same program, the executable part of the program is loaded only once into memory. If you deploy a new version of the program while users are still connected, nothing is broken and users continue to work with their current version. However, if they simply sign off and sign on again, they have the new version of the program. For a 5250 environment, this dynamic deployment is not possible because display file objects are allocated while they are open. In a multitier environment, this is entirely possible.

IBM i is scalable for performance and security is its strength. IBM i has had this capability for many years. Your applications can already take advantage of the cloud.



Modern development tools

There are many categories of development tools. This chapter covers some of the tools that are commonly used by RPG and COBOL developers. It makes sense that an application modernization project might also include modernization of your development toolset.

The category of tools that applies to most shops relates to editing, compiling, and debugging host applications, which are sometimes part of a set of tools that are called an integrated development environment (IDE). In addition, there are tools that are related to source code change management or source control tools. This chapter looks at both of those categories of tools and a few other tools that might be useful for specific environments.

The changes to your application might be simple technically, but there are also other aspects of a modernization project to consider:

- ▶ **Plurality of culture:** Developers who are using different technologies need to share projects, information, and processes.
- ▶ **Plurality of technologies:** Interfaces and connections introduce cross-platform function dependencies.
- ▶ **Complexities:** New technologies and methodologies can introduce complexities that your existing tools fall short of being able to handle in a meaningful and reliable way.

This chapter describes the situation of and recommends changes to your development tools that might be required for your modernization project. It also describes how to change your development methodology when working with these tools.

4.1 Editing, compiling, and debugging IDE

The primary tool from IBM that is available in the edit, compile, and debug category is an IDE called IBM Rational Developer for i, RPG & COBOL Tools. Previous versions of this Eclipse-based IDE from IBM were known as IBM Rational Developer for Power Systems (RDP) and IBM WebSphere Development Studio Client (WDSC). The primary basic capabilities of these tools have stayed the same throughout the various versions. However, new features and functions were added through the years, including support for current language features in the editor and a graphical DDS window and report designer.

4.2 SEU and PDM to Rational Developer for i: Why change

Many IBM i developers have been using the same tool for editing, compiling, and debugging their code for decades. This tool is the “green screen”-based Application Development Tool Set (ADTS), which includes Programming Development Manager (PDM), Source Entry Utility (SEU), Screen Design Aid (SDA), and other tools.

ADTS is no longer the preferred tool for modern IBM i developers. The green screen tools have reached the limit of their capabilities and their functions have not been enhanced in many years. The last update to these tools was April 2008 when IBM i 6.1 was released. Since that time, the ADTS tool set stabilized. There are no plans to include any additional functions to these tools, including updates to the syntax verifiers for the supported languages.

The modern replacement development tools from IBM are currently based in the Rational Developer for i, an Eclipse-based IDE. Developers who are comfortable using the PDM/SEU toolset might wonder why they should change.

The following sections describe some reasons to change.

4.2.1 Current (and future) language support

IBM is no longer updating the older style ADTS tools for new language features. It has been many years since there was any functional enhancement to the ADTS tools. Until IBM i 6.1, IBM included support for new language features. For example, syntax checkers in the SEU editor were updated to understand new RPG keywords, operation codes, or built-in functions in each release.

The updates for new language features are no longer provided. Therefore, when using SEU to edit RPG or COBOL source, new language features for IBM i 7.1 and later are not recognized. This situation means that many of the exciting new free-form features for RPG are impossible to implement for developers using the ADTS tools. SEU users are already missing editor support for features such as the %ScanRpl (scan and replace) built-in function, RPG Open Access Handlers, and handling stored procedure result sets. Only the Rational Developer for i tool provides the latest in language support, both now and in future releases.

4.2.2 Eclipse-based development environment

The Rational Developer tools are based on Eclipse, an open source development environment for many languages on many different platforms. The Eclipse base provides many advantages to IBM i developers:

- ▶ The same skill set that you use when you develop for RPG or COBOL applications on IBM i can be used in many other environments because such a wide variety of tools are Eclipse-based. When you are developing web pages, writing or using code that is written in PHP or Java, or when you access many database-oriented tools, you can use many of the same techniques and skills. Because there are many tools that are based on the same core, you get predictability when you need to use these tools.
- ▶ Many plug-ins are available for Eclipse because it is so widely used, which means that you do not need to rely only on IBM to supply all the new functions that you might want or need. Someone else might have already written the function you want and made it available publicly. Some of these plug-ins might be specific to the IBM i platform. Examples of these include RPG /Free source conversions, 5250 emulators, message file editors, and other useful tools. There are additional plug-ins that are not specifically written for an IBM i environment but can still be useful, including many database-oriented tools. Often, these tools are available either for no cost or at a low cost.

4.2.3 Productivity features of Rational Developer for i

Perhaps the most important reason to switch from PDM/SEU to Rational Developer for i is the dramatic productivity gain that the more advanced tools offer. It would require a whole book to produce a full list of productivity features that Rational Developer for i offers over the PDM and SEU tools.

The following section highlights some of the most obvious productivity features of the Rational Developer for i source code editor specifically, along with some tools that are closely integrated with the editor that are designed to support the editing process.

Rational Developer for i graphical debugger

The Rational Developer for i debugger can monitor the values of selected variables while stepping through the code or stopping at breakpoints. The debugger also can contain both active (enabled) and inactive (disabled) breakpoints. This feature is valuable because the debugger can also remember breakpoints across multiple debug sessions if a problematic bug requires several days to correct.

An example of the Rational Developer for i debugger in action is shown in Figure 4-1. This screen capture illustrates several features of the Rational Developer for i debugger. The Monitors view (upper right pane) shows variable values that were chosen to be watched throughout the debugging session. Values that have changed since the last step or breakpoint appear in red. Other variable values can be shown by hovering over a variable to see its current value, as illustrated. Breakpoints are shown with a mark to the left of the statement number.

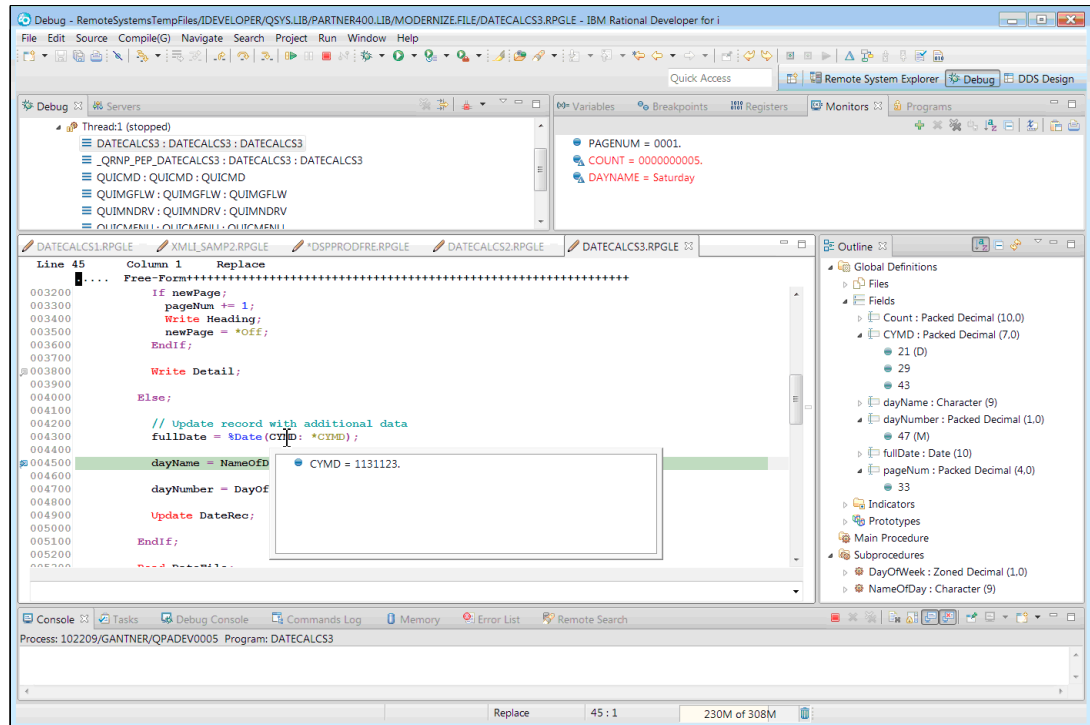


Figure 4-1 Rational Developer for i graphical debugger

DDS designer

In addition to the debugger, the DDS designer can make maintaining or creating reports or screens easier. The graphical DDS designer is the same tool for both reports and screens, unlike the SDA and RLU tools in the older ADTS toolset. The designer makes it far easier to move fields and text around and to center or align items. Fields from database files or tables can be chosen from a list and dropped in to the window to create database reference fields. It is simple to switch between graphical design mode and direct editing of the DDS for those occasions where the direct edit approach is more productive.

Figure 4-2 on page 49 provides an example of how the DDS design tool in Rational Developer for i works. This example shows a display file, but the interface for printer files is similar. The design pane in the center allows the developer to move or change the length of any items in the window by using drag or stretching. The Properties pane below the design window allows all the properties of the selected item (PRODCODE in this example) to be viewed and edited, including keywords and indicator conditioning (using other tabs in the properties window.) The Palette to the right of the design pane allows new items to be placed in the window by drag and drop. The Outline pane (lower left pane) shows details of the items in the window, including many keywords, and can also be used to select items in the design pane. At the bottom of the design window are three tabs, one of which (Source) allows the DDS source to be edited directly.

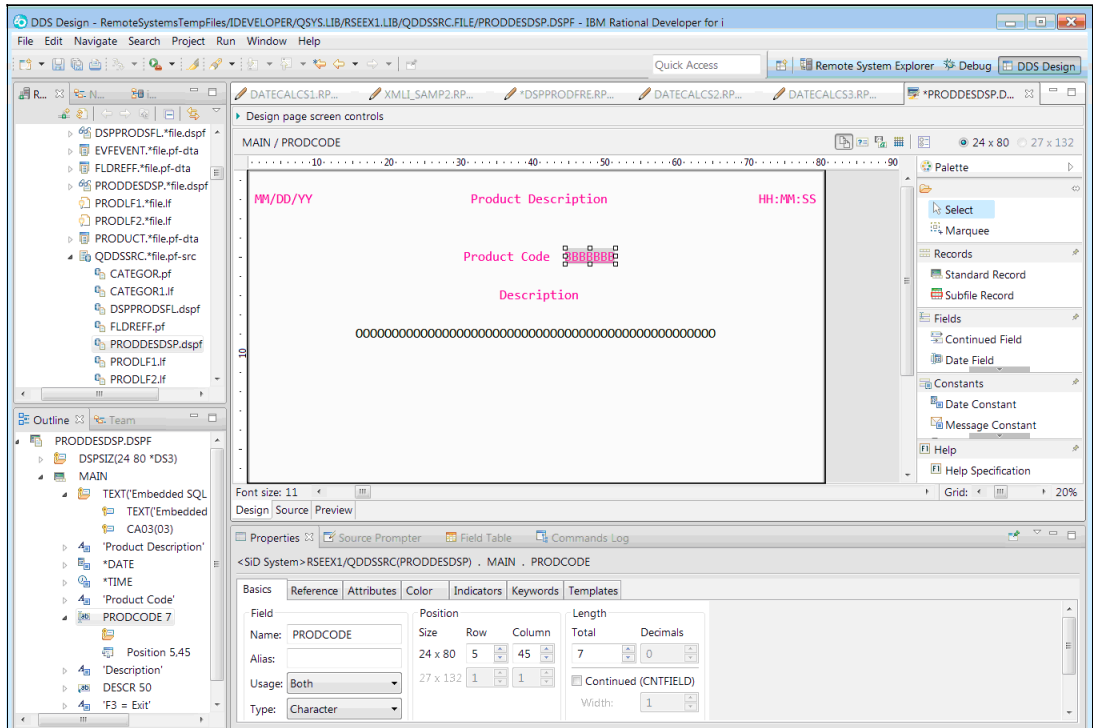


Figure 4-2 Rational Developer for i DDS design window and report editor

iProjects

There are other tools in Rational Developer for i, most notably iProjects, that support working in a disconnected environment. iProjects can also be used to help organize development work into local projects that are stored either on the workstation disk or on IBM i or another server. iProjects is also used with some source change management tools.

Rational Developer for i editor productivity highlights

Because most developers spend more of their time viewing, navigating, and editing source code, the productivity benefits of the Rational Developer for i editor deserve a more detailed look. The editor contains many features to enhance programming productivity. In addition to the editor itself, there are tools, such as the Outline and Error Feedback views, that are designed to work closely with the editor to further enhance productivity. This section highlights a few of the most valuable productivity features of the editor:

- ▶ More code visible at a time

When editing source code, a developer can typically see 2 - 3 times the number of lines of code compared to the older SEU editor. When using SEU, the number of lines visible is fixed, regardless of the size of monitor used. With the Rational Developer for i editor, the number of source lines visible varies based on the monitor size and shape and the font size selected. Seeing more code makes it easier and faster to follow the flow of logic. Figure 4-3 illustrates the ability to see more lines of code at once.

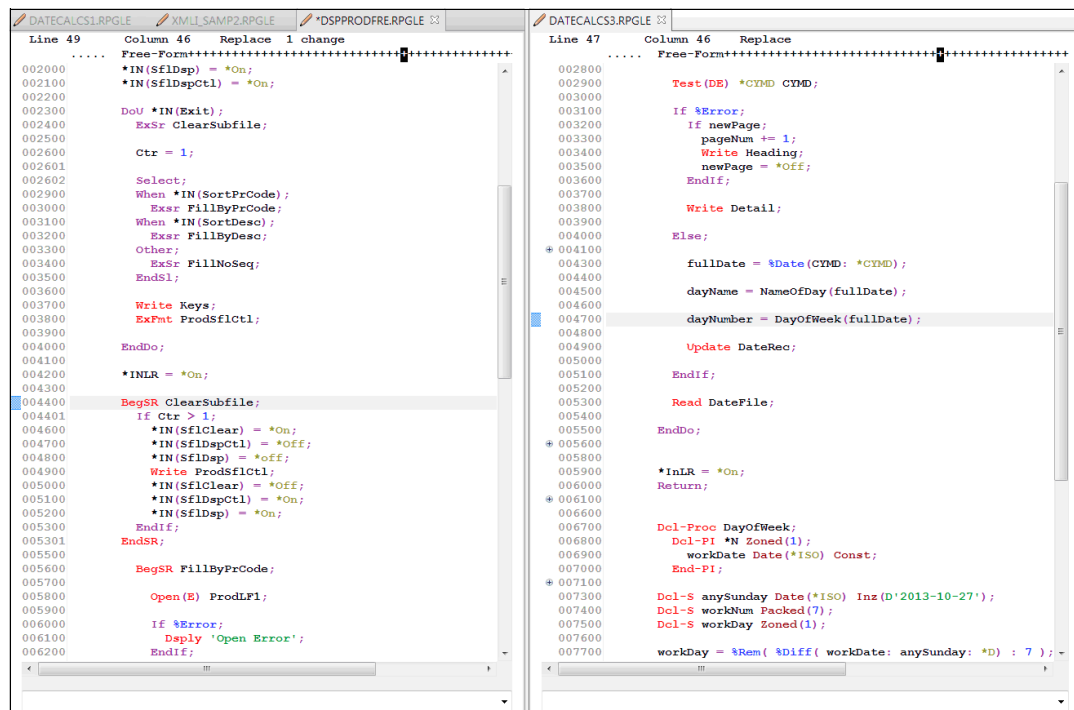


Figure 4-3 Rational Developer for i showing multiple programs in a split screen view

- ▶ Multiple source members open at once

Multiple source members can be open for edit (and browsing) at the same time. This ability becomes more important as modern applications become more modular. It is simple to click between the open members as the logic flows between modules. Two or more members can be seen simultaneously through one or more horizontal and vertical split screen lines. Even when in split screen mode, all members can be open for edit. Figure 4-3 shows an example of split screen editing in Rational Developer for i. In that same figure, the tabs showing above the source code indicate the different source members that are open in the edit session.

- ▶ Filtering of source code

Source code lines can be filtered in the Rational Developer for i editor in many ways. If there are many commented-out lines of code in a source member, choose to filter to see code only. This removes all the commented lines from your view, giving even more lines of code that are visible to the developer. Lines may also be filtered by the last change date on the line, which can be helpful when you look for recent changes to code that might have caused recent bugs within the application. Other filter options include showing only SQL statements, showing only subroutines or subprocedures, and showing only control flow logic statements. In addition, some text, for example, a field name, can be selected and then the code can be filtered on that selection. This shows only the lines of code referring to that field in the editor.

Filtering can help the developer find parts of a program that need attention faster.

- ▶ Outline view for definitions, cross-reference, and source navigation

A program Outline view is available to help you with navigation through the source code. The outline of an RPG IV program includes details of every file declared, including all record formats and all fields and their definitions. Details of all program-defined variables and data structures are also included. Both program-described and externally described variables appear in a list that can be sorted alphabetically by field name, which means that you quickly can find the field definition that is needed. Even without using the Outline view, the Rational Developer for i editor shows the definition of a variable when you hover the mouse over its name. So, you do not need to use a Display File Field Definitions (DSPFFD) or some other method on the host to find the definition of a field.

In addition to data definitions, the RPG IV Outline contains all subroutines, subprocedures, and prototypes that are defined in the source member.

The Outline is integrated with the editor, which means that a developer can navigate directly to the place in the program where a variable or subroutine or other item is defined or coded. In addition, there is a cross-reference in the Outline showing where each item is referenced in the source member; those cross-reference lines are also connected to the editor. So, for example, a developer can easily navigate not only to a specific subroutine using the Outline, but they can also navigate directly to every line of code in the member that calls the subroutine. The cross-reference information for fields indicates whether the field is modified at that statement by placing an (M) after the statement number. The Outline view is automatically updated as you edit code in the edit view. These two views are tightly tied together.

Figure 4-4 shows the Outline to the right of the code in the editor. Details of every line of code that references the fullDate variable, along with the definition of fullDate, is shown. Two of the statements referencing fullDate modify the value while one of them only references it. One of the references is selected in the Outline, which automatically positioned the editor to that line of code.

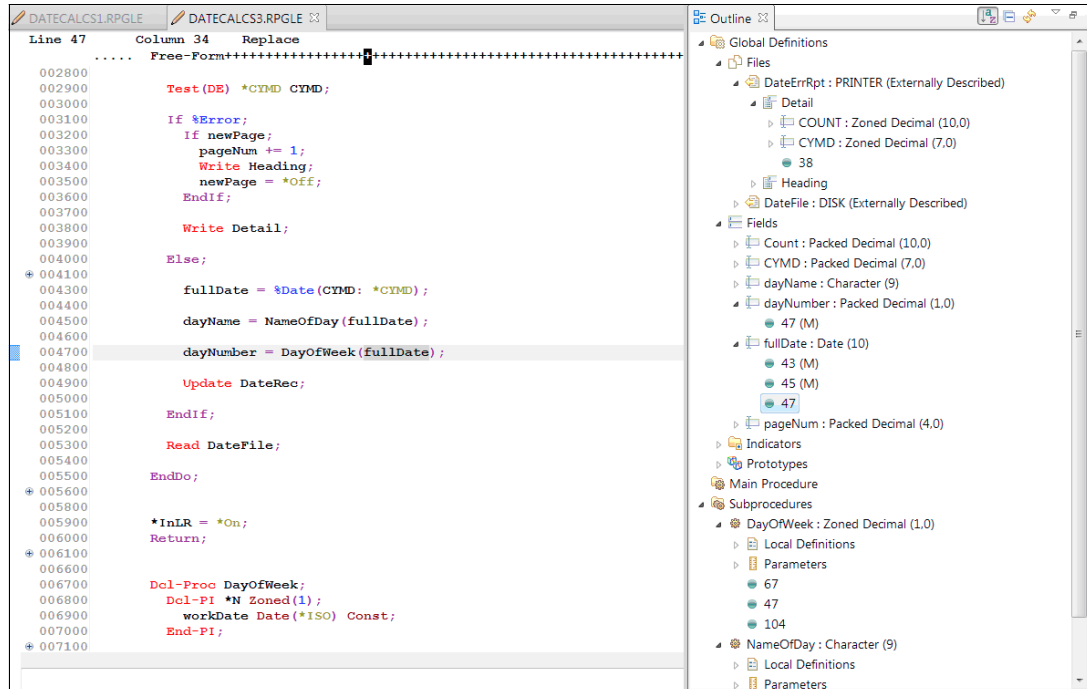


Figure 4-4 Rational Developer for i Outline view

- ▶ Undo and Redo

During an edit session, a developer has unlimited levels of both undo and redo of individual changes, even after a save and compile, while the source member is still open in the editor. This means that you can open a file, make changes, save, and then compile and do this sequence multiple times and still “undo” individual changes all the way back to when you opened that file. In the SEU editor, the only option to undo changes is to undo all the changes in the entire editing session by closing the editor without saving the member.

- ▶ Tab keys work in column-sensitive languages

For some column-sensitive languages, the tab keys work without the need for prompting. The tab positions can be customized to the developer's preferences. A format line appearing at the top of the editor window indicates the current position of the cursor on the line. Unlike the format line in SEU, which always reflects the format of the first statement on the panel, the Rational Developer for i editor changes the format line to match the statement where the cursor is positioned. This makes it much faster to enter code that is still in fixed-format RPG and DDS code.

- ▶ Error feedback that is integrated with the editor

Errors that occur during a compile appear in an Error list that is adjacent to the source editor. Double-clicking an error in the list positions you at the line of code in the editor where the error was found and places the error messages in the source. Developers never need to look at a separate spool file compile listings to find and fix compile-time errors in their code. Source members that are edited in Rational Developer for i are typically compiled without closing the source member to enable faster reaction to any compile-time errors.

In Figure 4-5, a compile attempt resulted in several errors. At the time of this screen capture, the developer double-clicked one of the errors and the editor was automatically positioned at the line of code for that error.

The integration of the error feedback with the editor is considerably faster than any mechanism that involves browsing of a spooled file compile listing to find lines of code in error.

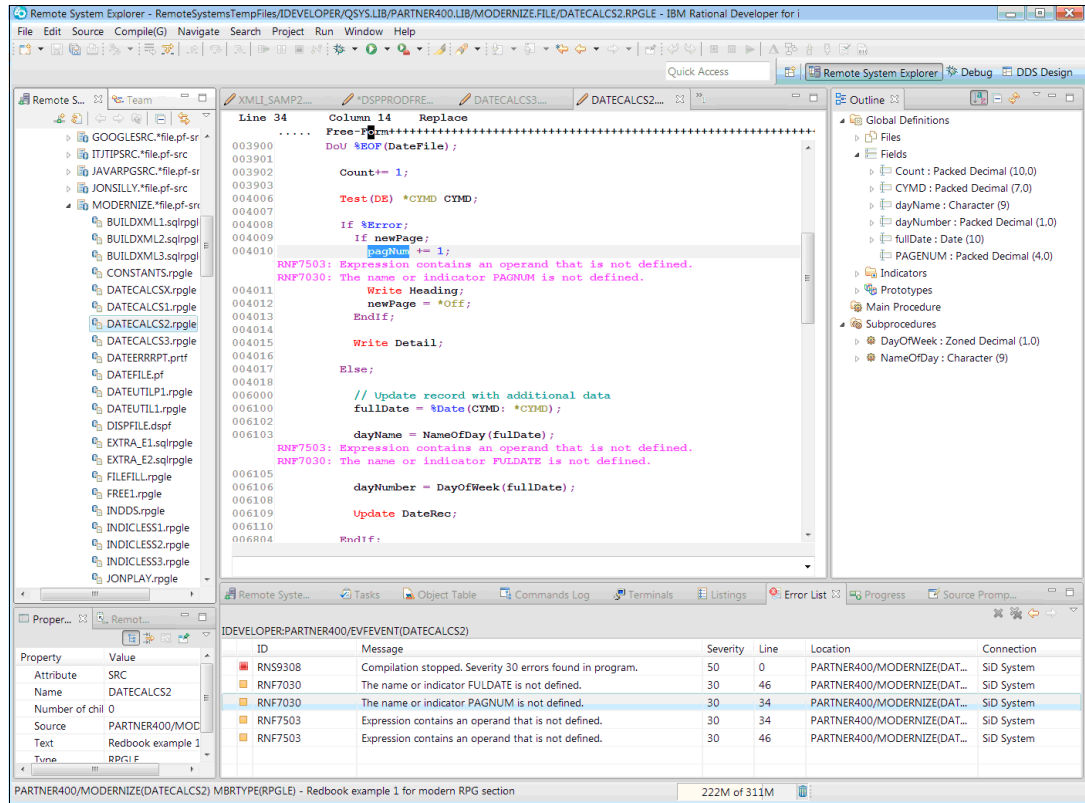


Figure 4-5 Rational Developer for i compile error listing and automatic positioning

This is a sample of some of the key productivity enhancements. As you start to use and learn this tool, you find your own favorite productivity enhancements.

4.2.4 Moving from PDM or SEU to Rational Developer for i

If you are a PDM or SEU user who is considering a move to Rational Developer for i, you can download a trial of the software that works for 60 days from the date of installation at the following website:

<https://www.ibm.com/developerworks/downloads/r/rdi/>

After the download, install the software on your Windows workstation (or in a Windows virtual machine on other platforms, such as Mac OS X). It is also supported on some specific Linux environments.

The following website can help you install Rational Developer for i on your system:

<http://iprodeveloper.com/blog/modern-i-zation>

To help you through some of the first steps of installation, updating the software, and making your first connection to your IBM i system, you might want to download the *RSE Quick Start Guide*, found at:

<http://systemideveloper.com/downloads.html>

In addition, after Rational Developer for i is installed, click **Help** → **IBM i RSE Getting Started**. Additional guidance to help get you started with Rational Developer for i is available at the IBM developerWorks® website:

<https://www.ibm.com/developerworks/downloads/r/rdi/support.html>

It is important to develop a different set of habits when working with Rational Developer for i. For example, when editing code with SEU, the source member is typically opened to make changes and then closed to save those changes before submitting a compile request. If the compilation is unsuccessful, then the same source member must be opened again to find and fix the errors. When using the Rational Developer for i editor, keeping the source member open when submitting a compile request enables the use of the much faster integrated error feedback tool. Likewise, in SEU it is often necessary to close a source member to open another one for edit. In Rational Developer for i, as many source members as are needed can be opened at the same time. Clicking with a mouse (or using a keyboard shortcut) allows switching between open source members quickly and easily.

Learn to use the strengths of Rational Developer for i to your advantage. Education is important, whether it is a hands-on instructor-led class, an online course, self-study, or a combination of these methods. Look for education that might be available to help you make the transition faster. A good reference book, even though it is a few years out of date as of this writing, is *The Remote System Explorer* by Yantzi, et al, which can be found at:

<http://www.mc-store.com/The-Remote-System-Explorer-Yantzi/dp/1583470816>

There are online communities that can help your transition as well. The WDSCI-L mailing list on <http://www.midrange.com> is an excellent resource. You can find the archives at the following website:

<http://archive.midrange.com/wdsci-1/index.htm>

You can join the mailing list to submit your own questions. There is also a forum on IBM DeveloperWorks, found at the following website:

<https://www.ibm.com/developerworks/community/forums/html/forum?id=11111111-0000-0000-0000-00000002285>

Many articles have been written on this subject. A few are listed here, but you can find many more with the help of your favorite internet search engine. The product name has changed a few times, from WDSC to RDP to Rational Developer for i, but most of the information is likely to still be valid even if the articles refer to an older version of the toolset.

Education is an important part of quickly making progress on learning anything new. There are a couple of key places you can get instructor-led education:

- ▶ IBM Lab Services

http://www.ibm.com/systems/services/labservices/platforms/labservices_power.html

- ▶ Partner400

<http://www.partner400.com/>

The following list provides references for additional education:

- ▶ Three Top FAQs for RSE:
<http://bit.ly/11YiH0u>
- ▶ Using Screen Designer:
<http://www.ibmssystemsmag.com/ibmi/developer/websphere/Using-Screen-Designer/>
- ▶ The Why and How of Moving from SEU to RSE:
<http://bit.ly/1oEgvhp>
- ▶ 3 Little Words that Simplify Debugging:
<http://bit.ly/1kCK2Yv>
- ▶ My Favorite Keyboard Shortcuts for RSE:
<http://www.itjungle.com/fhg/fhg041807-story02.html>
- ▶ Tips for using RDP's DDS Designer:
<http://bit.ly/LVsLuu>
- ▶ Dealing with Library Lists in RSE/RDP:
<http://bit.ly/1grB9w0>

4.3 Other tools

There are many other tools that exist and can bring valuable help or complementary functions when you do modern development.

4.3.1 Application Management Toolset

Application Management Toolset (AMTS) from IBM Rational is a new tool from IBM. It is a subset of the Application Development Toolset (ADTS). AMTS addresses the needs of operators and administrators who need low-level object and source management. The old ADTS toolset was used by two types of people in IT:

- ▶ Developers
- ▶ Operators and administrators

First, developers need the high-level capabilities that are provided by a toolset such as Rational Developer for i. Operators and administrators have simpler needs, primarily related to native object management and simple Control Language (CL) member editing.

Although Rational Developer for i provides the support that developers need, AMTS meets the needs of operators and administrators. It includes two major components of ADTS:

- ▶ Programming Development Manager (PDM)
- ▶ Source Entry Utility (SEU)

The goal of AMTS is to provide IBM i system administrators (and others) with a lightweight set of tools for general system management tasks. It can be used to browse, move, filter, and manipulate objects of any type, but it enables only development options (such as Edit and Compile) for Control Language (CL) and commands.

It is a system and operation team-oriented product, instead of a development tool.

AMTS uses the same PDM and SEU interfaces that have been available for years, but the AMTS editor allows editing of only the following member types:

- ▶ CL
- ▶ CLLE
- ▶ CLP
- ▶ TXT
- ▶ CMD

It provides language-sensitive features, such as syntax checking and prompting for CL and CMD member types only.

You can browse, but cannot edit the source, from other types of languages, such as RPG, COBOL, C, C++, or DDS.

You use AMTS by entering commands that are similar to the ones that are used for PDM, replacing PDM with AMT:

- ▶ **STRAMT** (Start AMT)
- ▶ **WRKLIBAMT** (Work with Libraries using AMT)
- ▶ **WRKOBJAMT** (Work with Objects using AMT)
- ▶ **WRKMBRAMT** (Work with Members using AMT)

Figure 4-6 shows the main menu for AMTS.

```
Application Management Toolkit (AMT)

Select one of the following:

    1. Work with libraries
    2. Work with objects
    3. Work with members

    9. Work with user-defined options

Selection or command
====> _____

F3=Exit      F4=Prompt    F9=Retrieve    F10=Command entry
F12=Cancel   F18=Change defaults

(C) COPYRIGHT IBM CORP. 1981, 2007.
```

Figure 4-6 AMTS main menu (STRAMT command)

Search facilities are accessible through option 25 from the member management panel or the **FNDSTRAMT** (Find String with AMT) command.

The command to start the Editor is **EDTCLU**, which is also used for option 2 from the **WRKMBRAMT** panel.

Other tools from ADTS, such as Screen Design Aid (SDA), Report Layout Utility (RLU), and Date File Utility (DFU), are not included in AMTS.

Figure 4-7 on page 57 illustrates the error that occurs when an AMT user attempts to open for edit a source member type that is not supported by AMTS. Option 5 displays the source works for this case.

```

Work with Members Using AMT                                     ITSOP1
File . . . . . QDSSRCD
Library . . . . . FLGHT400           Position to . . . . .

Type options, press Enter.
2=Edit      3=Copy  4=Delete 5=Display      6=Print      7=Rename
8=Display description 9=Save 13=Change text 14=Compile 15=Create module...

Opt Member      Type      Text
 2 FRS00ZDF      DSPF      Agent Logon thru USRPRF
   FRS000DF      DSPF      Display Flights Logon
   FRS001DF      DSPF      Flight Reservation data entry - new order
   FRS002DF      DSPF      Flight Reservation data entry - update order
   FRS003DF      DSPF      Flight Reservation - display order
   FRS004DF      DSPF      Flight Reservation - delete order
   FRS005DF      DSPF      Flight Reservation - fax order
   FRS009DF      DSPF      Flight Reservation - Select Order
                                           More...

Parameters or command
==>
F3=Exit      F4=Prompt      F5=Refresh      F6=Create
F9=Retrieve   F10=Command entry F23=More options F24=More keys
EDTCLU command does not support the Member type DSPF

```

Figure 4-7 DTCLU refuses to open a DSPF member in Edit mode

The AMTS licensed program is 5761-AMT. It is a tier-based product, so you pay a single, affordable price per system (serial number), and you can put AMTS on as many partitions on that system as you want. There is also no limit to the number of users that can use AMTS on that partition.

AMTS is an important part of the process of moving the developer community to a more modern development environment, while still offering basic object management support and CL support in a “green screen” tool. This affordable tool helps system administrators and operation teams with scripting, CL/CMD and basic text, and basic object management needs, and frees your development dollars for more advanced developer tools. Using AMTS means you no longer need to keep additional user licenses for the AMDS tools for your system operators.

4.3.2 RPG Next Gen

RPG Next Gen Editor is a lightweight RPG editor that is based on the Eclipse platform. It includes some basic plug-ins, which among other things provides a tree view of the IBM i QSYS file system. It was developed to be platform independent and works on Windows, Linux, and Mac OSX.

The focus on the editor is its small size and speed. The goal of this project is a customized RPG free-format editor, which can be used to develop RPG applications in an Eclipse-based tool that is supported on multiple workstation platforms.

RPG Next Gen is an open source software product that is released under the GNU Library or Lesser General Public License version 3.0 (LGPLv3). It can be downloaded from the following website:

<http://rpgnextgen.sourceforge.net/>

RPG Next Gen provides an inexpensive and quick alternative to Rational Developer for i to do RPG development in an Eclipse environment. It does not have the functions of the Rational Developer for i editor and does not integrate with the Remote System Explorer in Rational Developer for i. It also might not integrate well with IBM i source management tools. However, its low price and support for platforms such as Mac OSX and Linux might make it an attractive option for some RPG developers.

4.3.3 RPGUnit

RPGUnit is an open source framework that is designed to make unit testing easy for RPG programmers.

RPGUnit is a regression testing framework, similar to JUnit but written in RPG. Developers use it to implement unit tests in RPG ILE.

RPGUnit is open source software product that is released under the Common Public License Version 1.0. You can find it at the following website:

<http://rpgunit.sourceforge.net/>

Here are some of its features:

- ▶ General assertion facilities.
- ▶ Automatic detection of test procedures using introspection.
- ▶ Special setup and teardown procedures to avoid duplication between test cases.
- ▶ Standard compiling command.
- ▶ CL friendly command-line runner. A failure or an error is sent to the calling program

RPGUnit allows you to use continuous integration and unitary regression testing. It can automate simple tests of your batch program and ILE service programs. Using continuous integration, you can speed up the notification of regression errors to the developer.

4.3.4 Zend Studio for IBM i

Zend Studio is an industry-leading PHP IDE. It includes all the development components that are necessary for the full PHP application lifecycle. Through a comprehensive set of editing, debugging, analysis, optimization, and database tools and testing, Zend Studio accelerates development cycles and simplifies complex projects.

Zend Studio is based on Eclipse like other modern IBM i Tools, such as Rational Developer for i. This eases the learning curve and enhances integration. It is a good choice for shops that are modernizing their IBM i applications with PHP technologies.

The Zend Studio for IBM i edition brings features that are customized for the IBM i environment. It gives developers access to IBM i functions and applications:

- ▶ Job management
- ▶ Direct connection (create/use a connection job)
- ▶ Data area management
- ▶ DataQ management
- ▶ Object management
- ▶ Program/Service program call (with I/O parameters)
- ▶ Spool management
- ▶ User space management

For more information about using PHP on IBM i, see Chapter 12, “PHP” on page 545.

Figure 4-8 on page 59 shows an example of Zend Studio. You can see the similarities to the Rational Developer for i development tools.

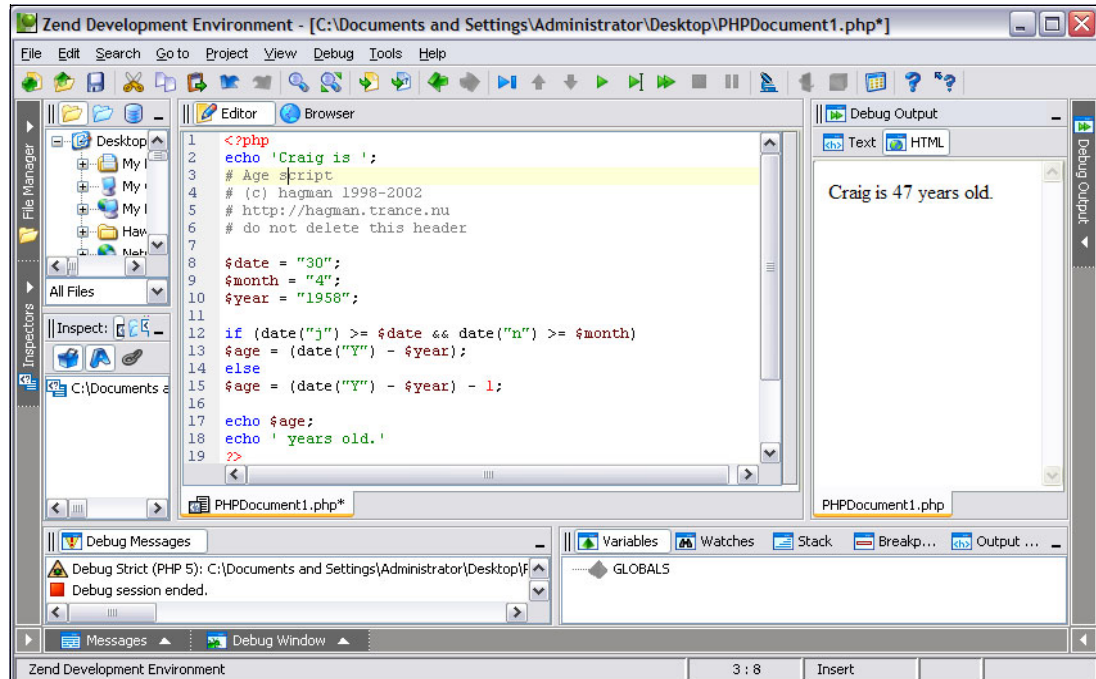


Figure 4-8 Zend Studio for PHP on IBM i

Here is a link to the Zend Studio (V10) website that explains all the features of this professional workbench:

<http://files.zend.com/help/Zend-Studio-10/zend-studio.htm#welcome.htm>

Zend Studio for IBM i can be downloaded from the Zend website:

<http://www.zend.com/en/solutions/i5-solutions/>

4.4 Source control and development management tools

There are two things to take into account for change management:

- ▶ Ongoing maintenance of existing applications
- ▶ New development for a modernization project

For legacy applications that are already in production, a tool to automate release management is helpful to ensure the control and the traceability of changes. In some shops, source change traceability is required by regulations in the business. Source change management tools can relieve developers of many small tasks and enhance their productivity. If you associate source control with a tracking system or a requirements management tool, you have a link between the expectations of users and the development team's work.

For the new development project that is related to modernization, the early stages of the project bring about different requirements for a change management tool. Figure 4-9 shows where the modernization project fits. A light source management system is a good answer to provide source security and traceability. It needs to allow the team to change their mind until a stabilized solution is provided. The tool must not slow the research and trial part of the project with strict control. During a development phase, only a part of what is experimented goes to production.

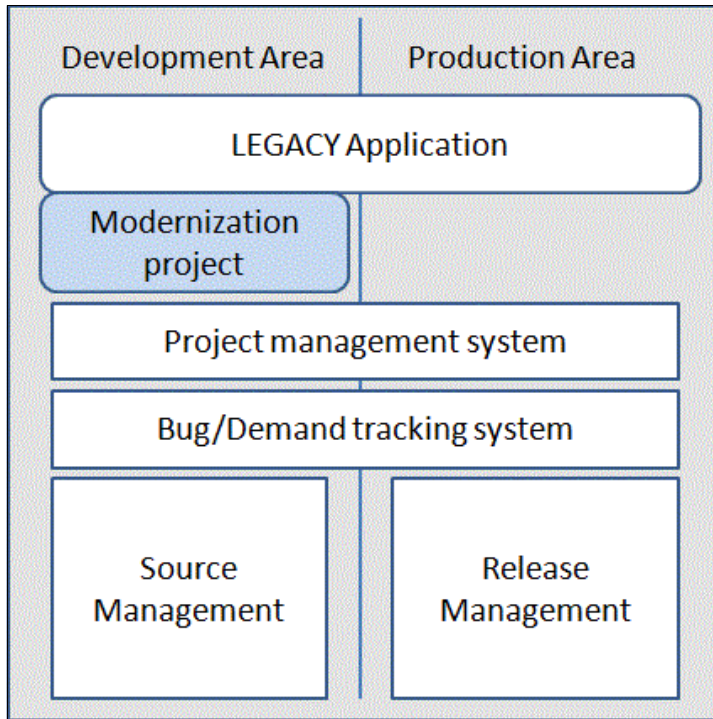


Figure 4-9 Positioning a modernization project inside the lifecycle

After entering production, the new development enters the maintenance phase and must enter regular change management phase, as shown in Figure 4-10 on page 61.

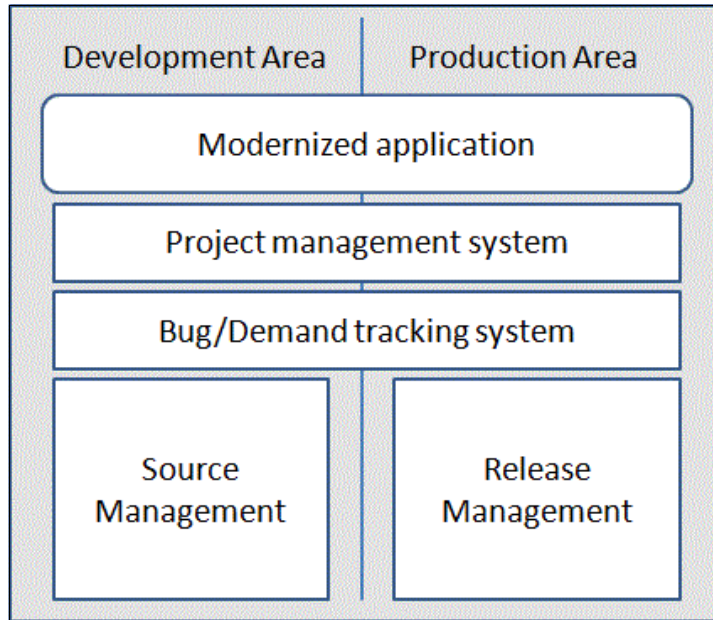


Figure 4-10 Modernized application in the last phase

The evaluation of the correct development management tools is important because it introduces multiple new technologies to your IT organization. You might not be using any management tools and this approach works for your existing support. Development might be managed by one group who has shared the same IT culture for years. This can be based on standards and an established manual methodology or “home-grown tools”, or one of the many change management tools that have existed for many years, and it might work well for the current environment.

However, modernization components might change the habits and introduce dependencies between components that might not be manageable by your current methodologies or by your current tools.

During a modernization project, business does not stop. Legacy applications are still moving to answer business demands. Tracking changes is a minimum requirement to ensure success and avoid loops in the change process.

The simplest minimal application lifecycle to conduct any transformation project requires two environments:

- ▶ One reference environment
- ▶ One development area

The reference environment is an image of the production area that is the base level of the application.

The development area is where the current changes might be and is often where developers “merge” their changes before delivering them to the reference environment and then to the production environment.

Figure 4-11 shows a simple view of a lifecycle.

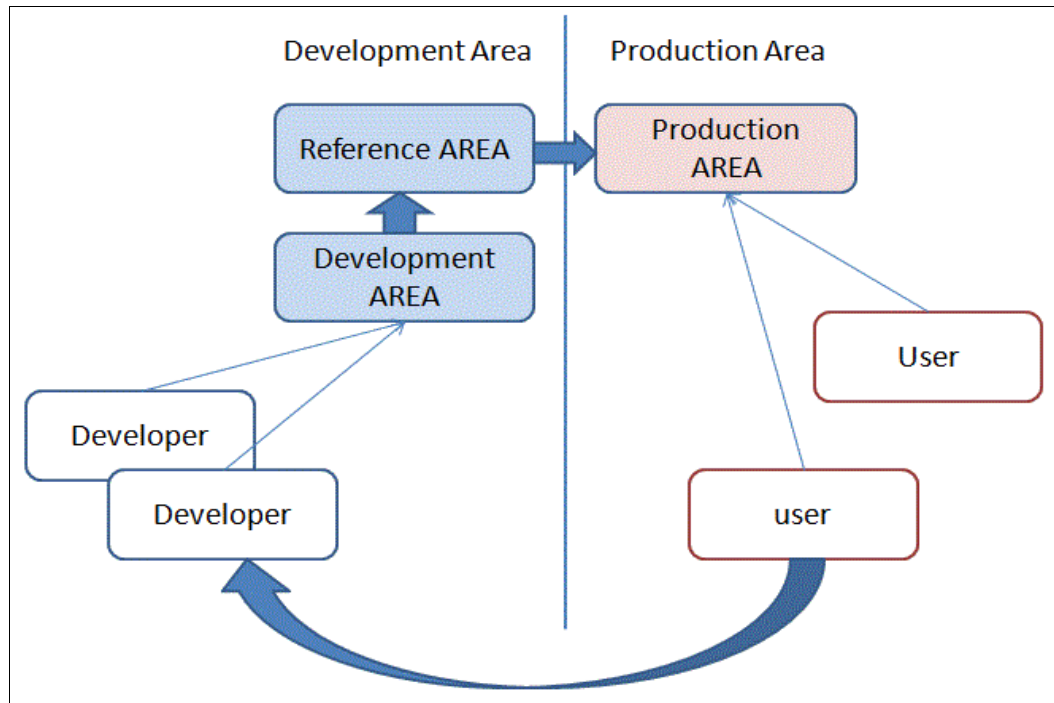


Figure 4-11 A simple lifecycle

This simple lifecycle works well for simple configurations that do sequential development (one change at a time). This is a *simple release management system*.

For more complex needs, such as parallel development, multi-level applications for multiple production environments, regulations, and quality assurance requirements, you need a more complex lifecycle.

Figure 4-12 on page 63 shows an example of a more advanced but common lifecycle.

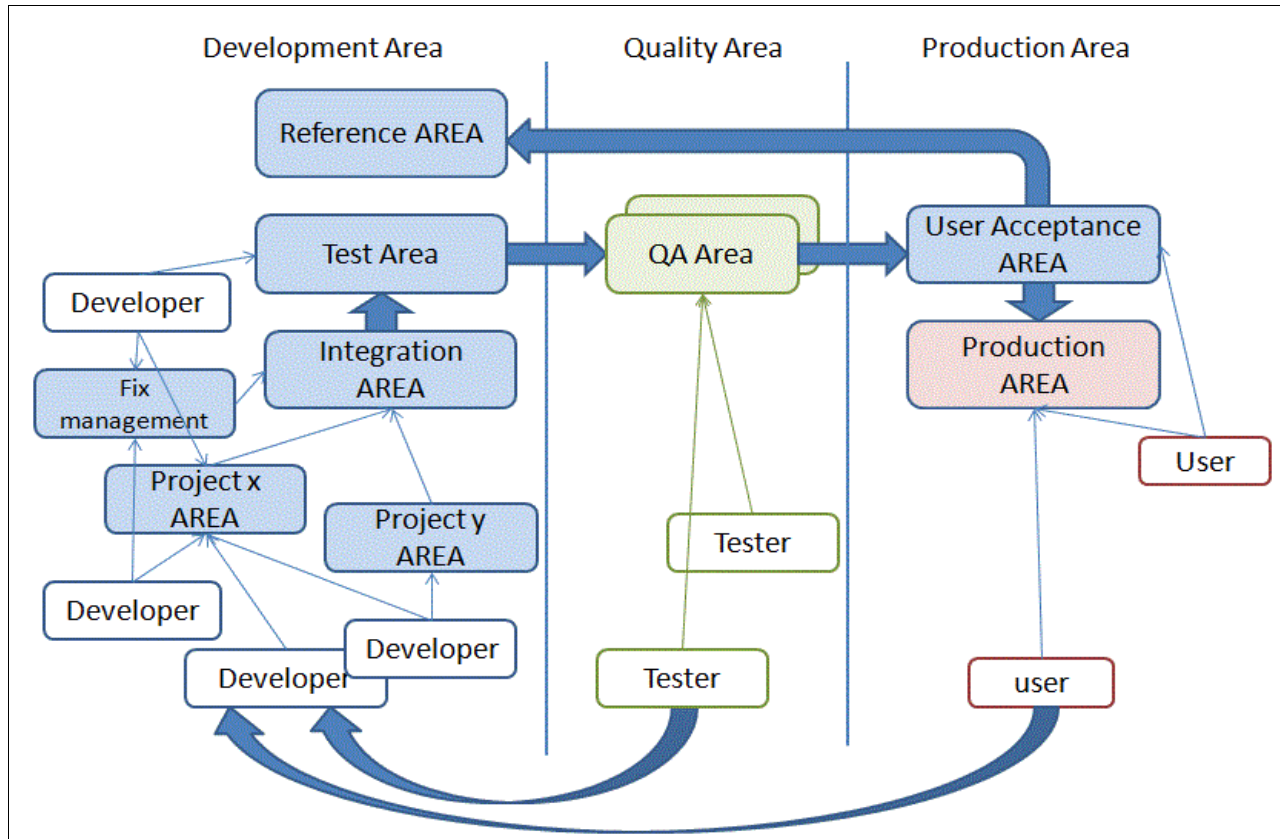


Figure 4-12 Lifecycle with quality assurance and user acceptance process

For either type of lifecycle, the lifecycle changes with your development organization. The most important criteria for implementing change management is that it allows the environment to change. It must serve the needs of your business and not become a constraint.

Carefully evaluate your choice for such a tool. Do a proof of concept (PoC) if you want to evaluate its suitability compared to your strategy. Here are some criteria to consider as you make your change management tool choice. Keep in mind that such a tool is not just an editor; it has implications for the entire IT organization.

Here are some criteria that can help you make the correct choice:

- ▶ Legacy applications must be taken into account (must be supported)
- ▶ Technologies that you want to use (most of them must be supported)
- ▶ Openness (it must be able to accept future technologies that do not exist yet)
- ▶ Teams involved (Cultures must be considered)
- ▶ Development methodologies (classical, agile (scrum ...))
- ▶ Lifecycle (simple to complex)
- ▶ Regulation and compliance pressures
- ▶ Ease of use and productivity

Consider also that in some cases more than one tool might be needed. Some functions must be shared by all teams:

- ▶ Project management
- ▶ Tracking system
- ▶ Source management
- ▶ Release management

IBM Rational Team Concert™ and ARCAD Pack for Rational is one of the more modern assemblies for development management tools. It has most of the items that are described above, including high-level support of IBM i technology. Section 4.5, “IBM Rational Team Concert and ARCAD Pack for Rational” on page 64 offers a description of IBM Rational Team Concert and ARCAD Pack for Rational.

4.5 IBM Rational Team Concert and ARCAD Pack for Rational

IBM Rational Team Concert for i is a collaborative software delivery environment that allows teams to simplify, automate, and govern application development on IBM i and distributed environments such as Java, PHP, or Microsoft.net. For a complete set of functions, from source tracking to source deployment on the IBM i platform, it can be combined with IBM Rational Developer for i and ARCAD Pack for Rational. Figure 4-13 illustrates the activity domain of each part of the solution.

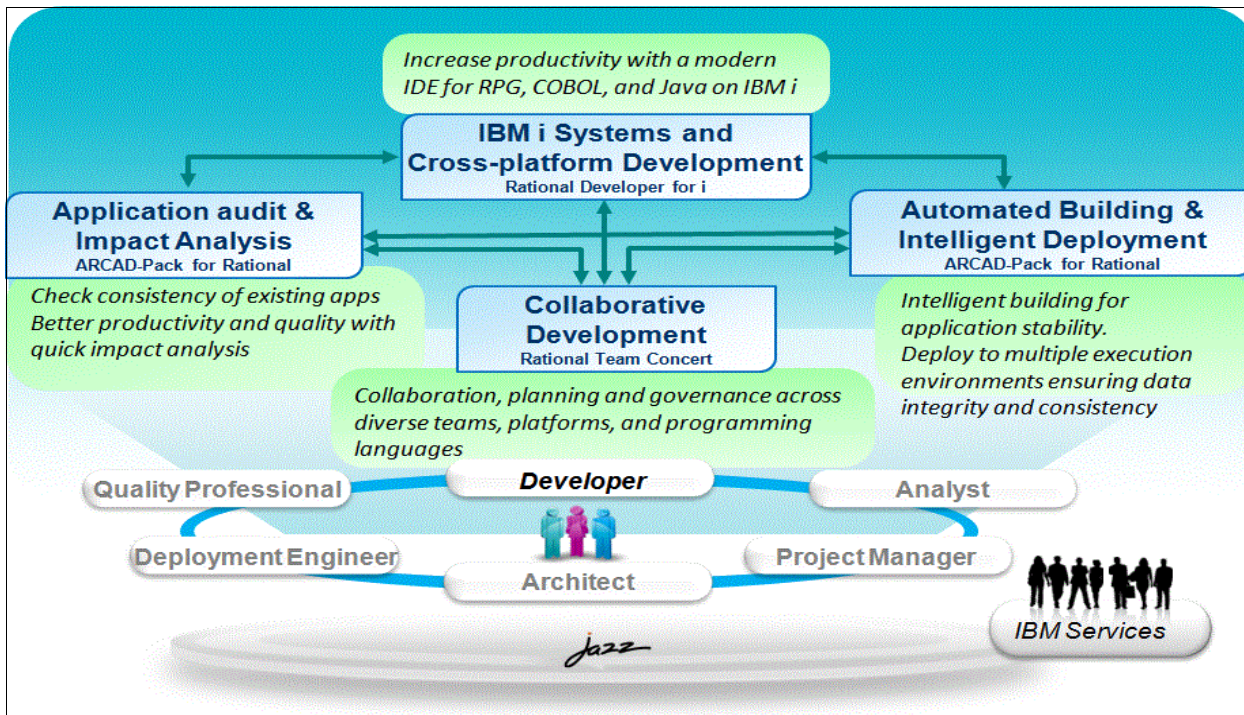


Figure 4-13 IBM Rational Team Concert, IBM Rational Developer for i, and ARCAD Pack for Rational connections

Rational Team Concert provides the following capabilities:

- ▶ An integrated set of collaborative software delivery tools for IBM i development, including source control, change management, build, process management, and governance
- ▶ Integration with IBM Rational Developer for i to enable team capabilities for native IBM applications
- ▶ Specialized support for source control, change management, and builds of traditional language artifacts, such as RPG and COBOL
- ▶ Support for multitier software development and application modernization efforts using RPG, COBOL, PHP, Java, and others
- ▶ Supports IBM i native library file system and integrated file system (IFS)
- ▶ IBM i artifact builds, including RPG, COBOL, CL, and Java

- ▶ A build agent, which runs natively on the IBM i operating system (runs IBM i commands and call programs)
- ▶ Native hosting of the IBM Jazz™ Team Server on IBM i

IBM Rational Team Concert for i provides these capabilities by extending the IBM Rational Team Concert product.

From the modernization point of view, IBM Rational Team Concert is an answer for developing traditional business applications while also modernizing existing applications using diverse, non-integrated development tools that isolate RPG/COBOL developers from web, Java, PHP, and .NET developers, yet allow these teams to work together toward a common goal by using common tools.

4.5.1 Description

Rational Team Concert for i is based on the open and extensible Jazz platform. It empowers software teams to collaborate in real time in the context of specific roles and processes by delivering new levels of productivity and agility.

IBM Rational Team Concert for i, combined with ARCAD Pack for Rational, integrates source control, work items, project tracking, high-level dependency build management, impact analysis, and automation in deployment into a single application. It enables distributed teams that are spread throughout your enterprise to more effectively collaborate in developing and delivering quality software on time and on budget for modern IBM i applications. It allows teams to simplify, automate, and govern application development on IBM i. IBM Rational Team Concert for i builds on the core IBM Rational Team Concert product by providing a Team Server and Build System Toolkit that runs on IBM i. The IBM Rational Team Concert for i client is shipped as an installable component of IBM Rational Developer for i. To use the IBM Rational Team Concert for i functions for team collaboration and managing RPG and COBOL applications, you must use IBM Rational Developer for i with IBM Rational Team Concert for i.

IBM Rational Developer for i is an integrated development environment for creating new and modernizing existing RPG and COBOL applications.

IBM Rational Team Concert for i includes the following integrated components:

- ▶ Software configuration management
 - IBM Rational Team Concert for i delivers essential software version control, workspace management, and parallel development support to individuals and teams. It includes the following specific functions:
 - Developer to team flow
 - Integrated stream management
 - Component-level baselines
 - Server-based sandboxes
 - Components that are identified in streams and available baselines
 - The capability to store, control and track changes to RPG, COBOL, and Java application source in a repository
 - Support for IBM i native library file system and the integrated file system (IFS)

► Work item management

IBM Rational Team Concert for i supports different types of work items for software development teams. Team members can discuss a particular work item and discussions are captured for later reference. Work item owners or interested parties can subscribe and receive notifications through RSS and Atom feeds. Team members can share queries with the team or with particular users. It includes the following functions:

- Defects, enhancements, and tasks
- Query results view and share queries with the team or members
- Support for approvals and discussions
- Tracking of changes to RPG, COBOL, and Java source through work items
- Query editor interface

► Build management with Extended Dependencies Build

IBM Rational Team Concert for i enables efficient scheduling and running of the software build process. You can use multiple servers for rapid, cross-platform build processing and create a detailed bill of materials to enhance build reproducibility. Here are additional functions:

- Work item and change set traceability
- Local or remote build servers running on IBM i
- Build for RPG and COBOL applications on IBM i
- Support for Ant and command-line tools for Java
- Build definitions for team builds
- ARCAD-Builder enhances the base build features to allow a dependency build to take care of IBM i specific object dependencies, such as file to program and ILE service program to modules and to programs

Figure 4-14 shows IBM Rational Team Concert combined with ARCAD-Builder to ensure a consistent build process for your IBM i objects.

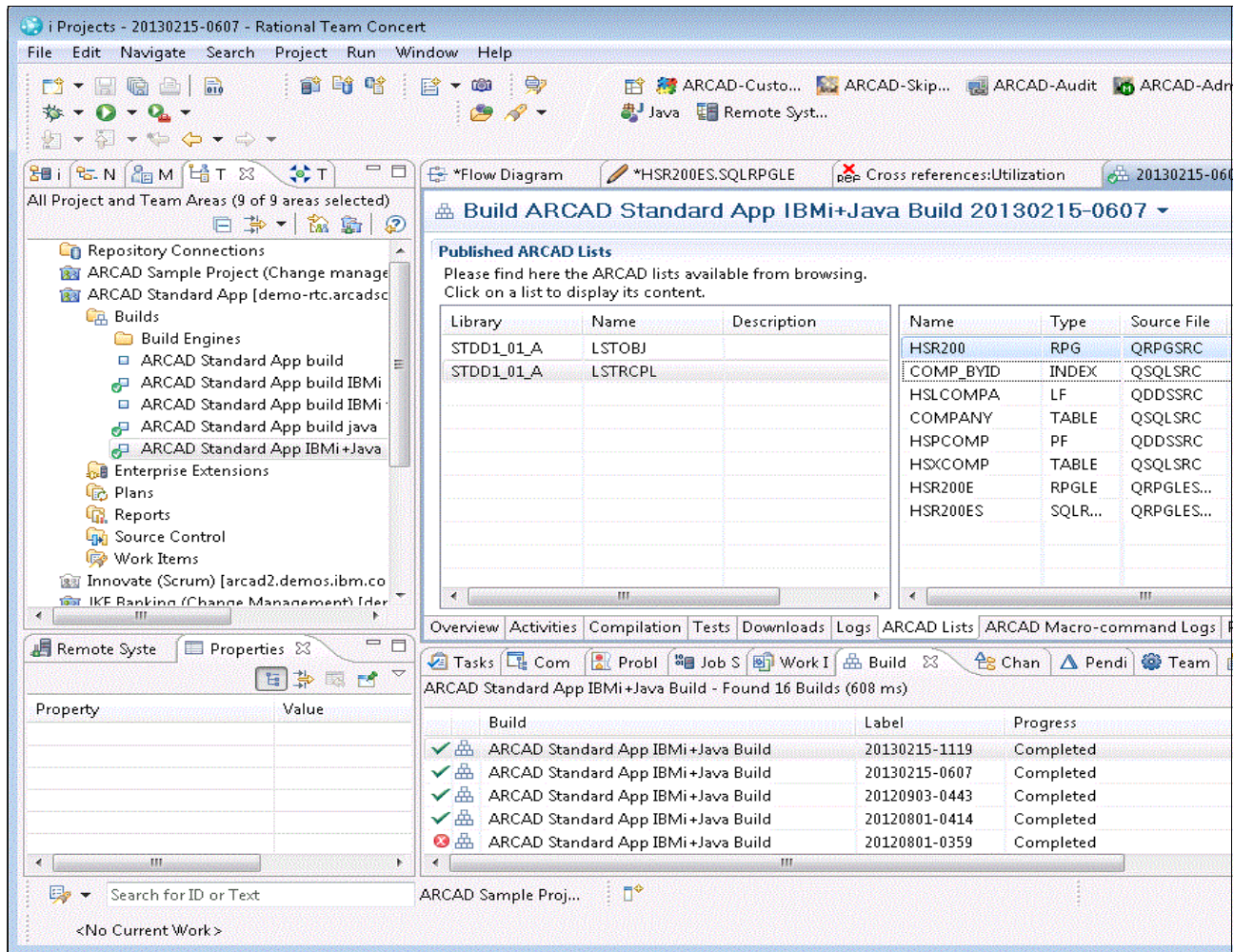


Figure 4-14 A build result sample that shows some recompilation because of changes

► Impact analysis:

Within the ARCAD Pack for Rational, IBM Rational Team Concert for i enables efficient impact analyses directly from the developer workbench. You can benefit from ARCAD Open repository and ARCAD-Observer impact analysis capability. Impact analysis queries are linked with change management to bring efficient query results regarding what a project user is working on. Figure 4-15 shows an impact analysis when combining the ARCAD-Observer with Rational Team Concert.

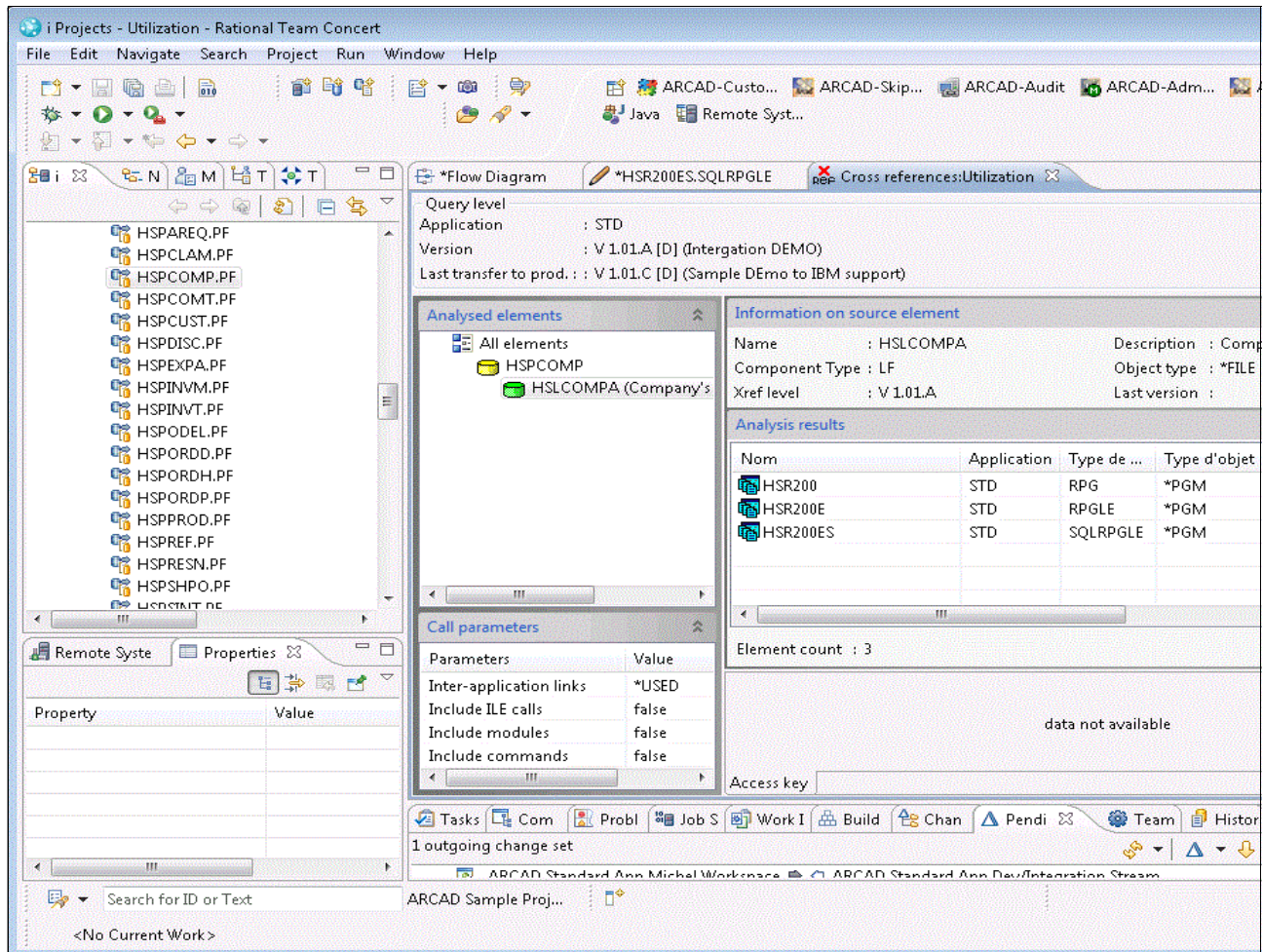


Figure 4-15 Result of an impact analysis from iProject

Here are its functions:

- Several repositories
 - Objects and Source
 - ILE procedures
 - Fields
 - Literals
- Multiple viewpoints
 - Applications
 - Functions (functional subdivision of an application)
 - Components (Source and Objects)
 - Artifacts
 - Source code lines

- Search features
- Multi-platform cross-references
 - All IBM i native languages
 - Java
 - PHP
 - Microsoft .NET
- ▶ Automated deployment on multiple platforms:

Leveraging the use of ARCAD-Deliver, one of the solutions that is included in ARCAD Pack for Rational, IBM Rational Team Concert for i automates and secures multiplatform deployment of modern applications to their production locations. With its openness to the common standard process managers, such as ANT, this Automation Agent is available on all platforms. It can be used with many technologies, such as Java, web servers, application servers, Windows servers, Linux, UNIX, and IBM i.

Here are its functions:

- Built-in IBM i process manager
- Pre-configured processes
- Automated transfer and compile features
- Management of IBM i specific attributes (Authorities/ownership/object attribute)
- Integrated smart data management (Optimize the use of ALTER/CHGPF)
- Transaction mode
- Rollback crossover platform

IBM Rational Team Concert for i helps you gain the cross-organizational visibility and collaboration you need:

- ▶ Process automation and guidance: Enforcement of agreed-upon standards can help ensure higher-quality results. Because not all organizations are the same, rules are configurable and can be defined or refined as needed, enabling continuous improvement following the rhythm of your modernization process.
- ▶ In-context collaboration: Through personal customizable views, team members can gain better understanding of what is happening on projects (news and events, build status, what is being worked on, and changes). Team members can also be made aware of what their teammates are working on and who is online and possibly available to collaborate with.
- ▶ Web client: Using Ajax technology, users experience a rich client style of interaction for web access. External stakeholders or occasional users can gain access to data without a rich client.

- ▶ Dashboards and reporting: Project, individual, and team dashboards offer reliable project health information that is drawn directly from ongoing work, replacing tedious and time-consuming reporting overhead. Figure 4-16 shows some of the diagrams that can be seen within a dashboard.

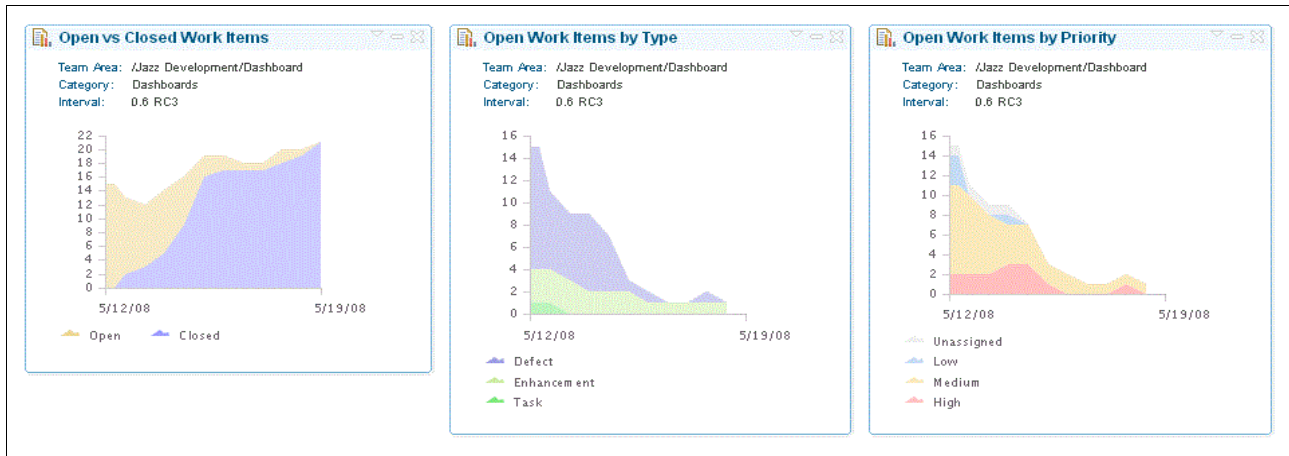


Figure 4-16 IBM Rational Team Concert work items diagrams that are extracted from a dashboard

IBM Rational Team Concert includes a complete set of integrated functions to enable teams to effectively deliver software on time and on budget:

- ▶ Integrated iteration planning and execution
- ▶ Real-time project status window
- ▶ Single structure for project-related artifacts
- ▶ World class team onboarding and offboarding, including team membership, sub teams, and project inheritance
- ▶ Role-based operational control for flexible definition of process and capabilities
- ▶ Team advisor for defining and refining rules and enabling continuous improvement
- ▶ Process enactment and enforcement
- ▶ In-context collaboration showing team members and work status
- ▶ Customizable project dashboard
- ▶ Real-time reporting and queries

As an integrated set of collaborative software delivery tools for IBM i development, including source control, change management, build, and process management, IBM Rational Team Concert for i and ARCAD Pack for Rational provide the capability to store, control, and track changes to RPG and COBOL application source in a repository, open work items, and associate changes to source with those work items and build your applications from the repository. IBM Rational Team Concert for i can be configured to reflect a team's process, including defining project iterations (milestones) and planning of each iteration, permissions, user roles, and assignment of each member of a team to a particular user role.

- ▶ Integration with IBM Rational Developer for i to enable team capabilities for native IBM i applications

IBM Rational Team Concert for i integrates with the iProjects component of IBM Rational Developer for i to provide an integrated user experience. Use the Eclipse-based environment to open or accept a work item (a task or a defect), check out an RPG, COBOL, DDS, or CL member, make changes to the member using the built-in LPEX Editor that provides support for syntax checking and color highlighting, verify or compile changes, and debug the application before finally checking in those changes to the repository and associating them to a work item. You can do all of these things seamlessly from a single client, which includes both IBM Rational Team Concert for i and IBM Rational Developer for i. Figure 4-17 shows the Rational Team Concert plug-in within the Rational Developer for i interface.

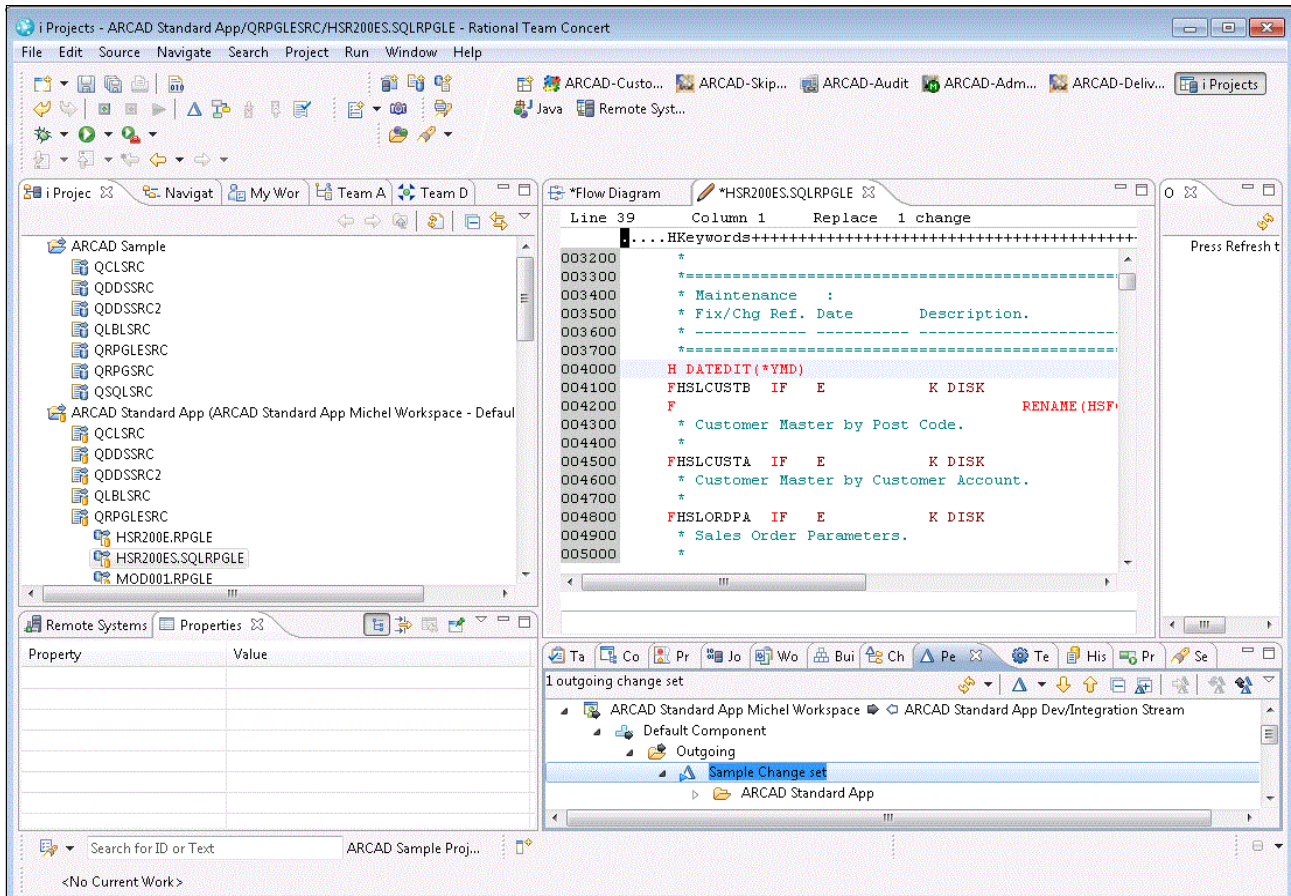


Figure 4-17 IBM Rational Developer for i integrated with Rational Team Concert

- ▶ Specialized support for traditional language artifacts, such as RPG and COBOL

IBM Rational Team Concert for i provides specialized support for source control and build of RPG and COBOL applications. This support includes a feature to automatically copy changes to the repository to the IBM i file system so that changes show up in team members' library lists. There is a view that shows comparisons between a repository and the IBM i library file system, and support for building traditional applications.

- ▶ IBM i artifact builds

IBM Rational Team Concert for i can be used to set up a build engine on any IBM i system that can extract source from the repository and run system commands (including compile commands or any customized build commands) to do integration, compile, and build. You can schedule a build to run at certain time intervals or at a certain time of day, or you can request a build on demand. The build can extract everything in the repository or just the changes that were made since the last build, and can run a particular command for every change. The commands allow substitution variables to be specified like Programming Development Manager (PDM).

ARCAD-Builder enhances IBM Rational Team Concert by adding processes that automate a dependency build to allow a smart regeneration of ILE components, automate object properties preservation, and automate data recovery.

- ▶ Support for multitier software development and application modernization efforts

You can use IBM Rational Team Concert for i for team collaboration on projects involving traditional applications using RPG and COBOL and involving Java. Rational Team Concert for i can be used to manage such multitier and application modernization efforts in a consistent way. The workflow and actions to check out, edit, compile and check in changes, and associate those changes with tasks or defects, is similar, whether the source is RPG, COBOL, DDS, CL, EGL, or Java.

Impact Analysis is embedded into the workbench and gives users a real-time global picture of multitier software that improves decision capability and reduces the risk of breaks between technologies.

4.5.2 Example of IBM Rational Team Concert configuration

Figure 4-18 on page 73 is a stream flow configuration example that shows a source management lifecycle with IBM Rational Team Concert. It includes three levels:

- ▶ Reference/production level
- ▶ Quality level
- ▶ Development level that has two streams
 - Current Development stream
 - Test fix stream

Each level has a repository workspace that is dedicated to a build and, at the bottom, there is a set of repository workspaces that are owned by developers.

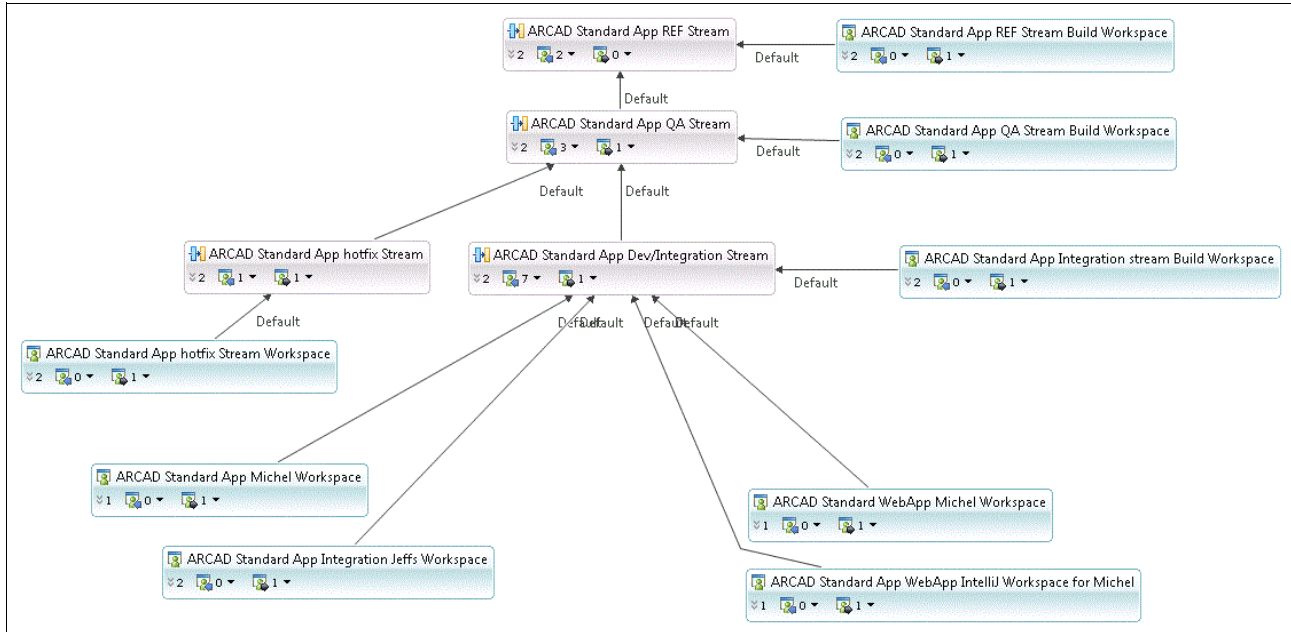


Figure 4-18 IBM Rational Team Concert flow diagram

4.5.3 Managing IBM i projects with IBM Rational Team Concert

The streams and repository workspaces in IBM Rational Team Concert allow you to have agile configuration management. It is easy for a specific project to add a stream, work as isolated as possible, and be merged into a regular stream when the project is ready. This stream creation does not duplicate any source code. Stream and workspace management is dynamic and includes snapshot management that is ideal to create a milestone picture every time you need it.

Figure 4-19 explains the concepts and actions of source management within IBM Rational Team Concert.

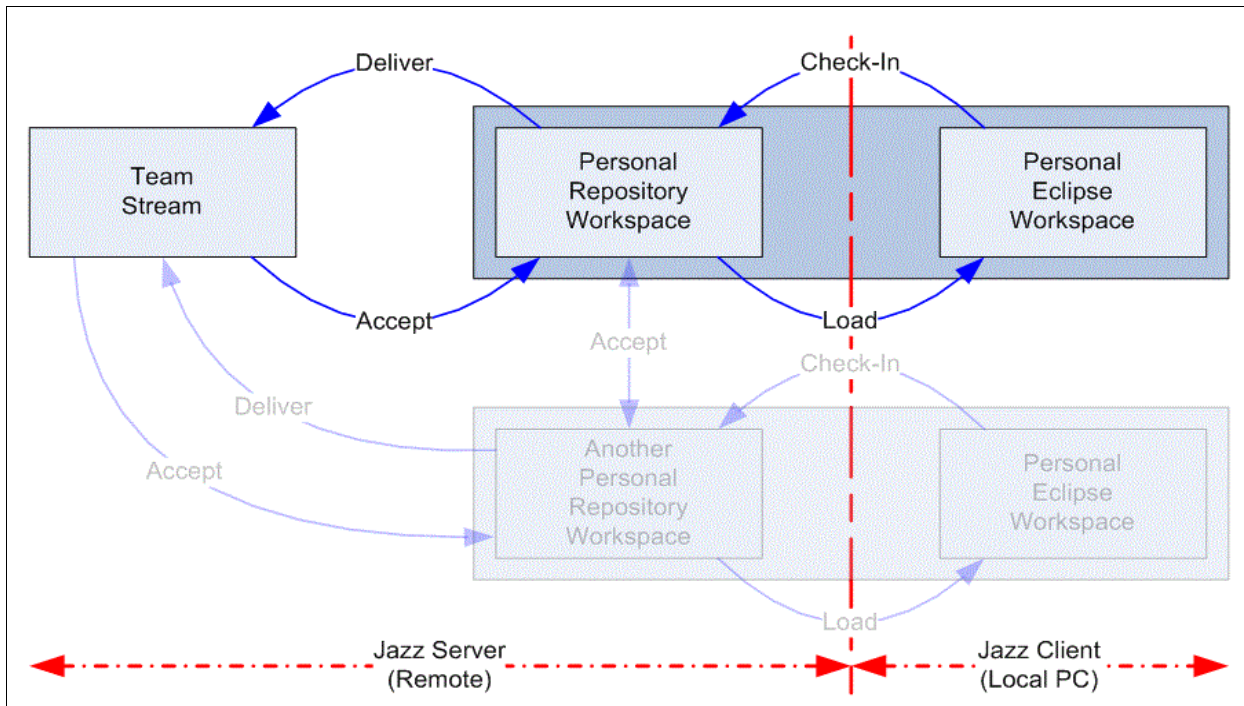


Figure 4-19 Example of the source check-in process

There are three types of spaces:

- ▶ Stream
- ▶ Repository workspace
- ▶ Local (Eclipse) workspace

There are four actions that manage exchanges between these spaces:

- ▶ Load

Load your repository workspace into your local workspace (this does a real source copy of the source).

- ▶ Check-in

Push the changed source back to the repository workspace (this does a real source copy).

- ▶ Accept

Accept the changes from another space. It might be a stream or another developer's repository workspace. (This does not copy source; it is a pointer.)

- ▶ Deliver

Deliver the changes to another space. It must be a stream.

These actions might require merge management in case there are source conflicts.

Powerful functions allow developers to navigate multiple projects at the same time if they must. A classic case is the fix in the middle of a project. The resume function puts the changes that you did to resynchronize to another level on a shelf. You can then do a change at this level and go back to working on the original project by resuming it. Figure 4-20 on page 75 shows this example.

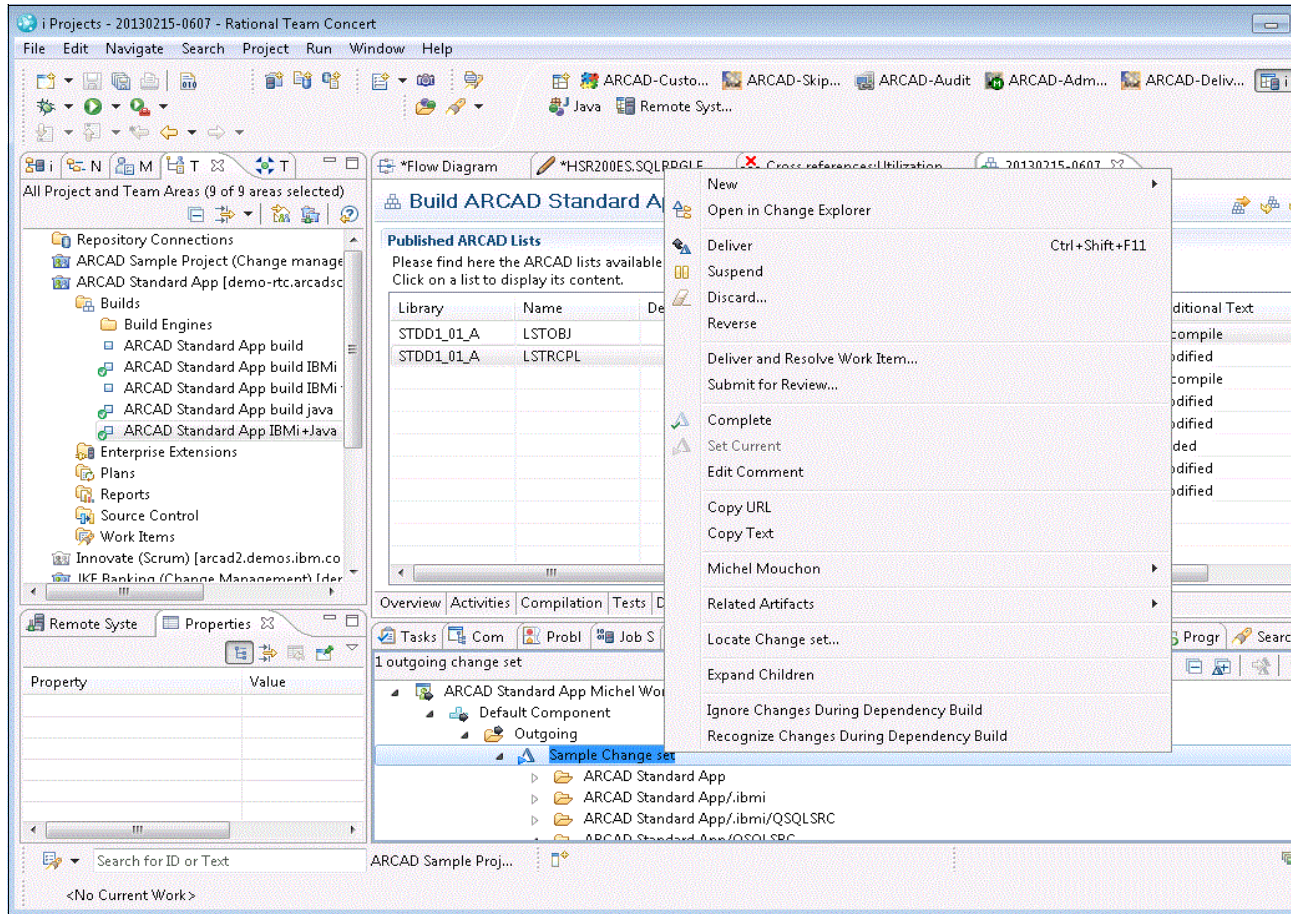


Figure 4-20 Changes are held until you resume them

As with any source configuration management tool, IBM Rational Team Concert does check-out and check-in even if this operation is transparent for the developer. By default, IBM Rational Team Concert use an optimistic check-in/check-out process to provide the maximum flexibility

Optimistic versus pessimistic

Optimistic management does not allocate the source to developers. So, any developer can change the same source when they must. The merge resolution is done when they deliver the changes from their repository.

Pessimistic management requests allocation of the source before a developer can change it. Some pessimistic source management tools do not allow parallel change of the same source, thus solving the merge issue by avoiding the merge. For flexibility reasons, pessimistic management has become obsolete.

There is also intermediate pessimistic- and optimistic-oriented management that allows parallel changes of the same source (such as ARCAD-Skipper). In this case, the pessimistic point of view takes a more proactive approach to merge management. The developers can take the source that is allocated and know that it must be managed in a future merge before starting the changes.

Figure 4-21 illustrates the possibility of parallel project management using a modern tool. This is inconceivable without a source management tool.

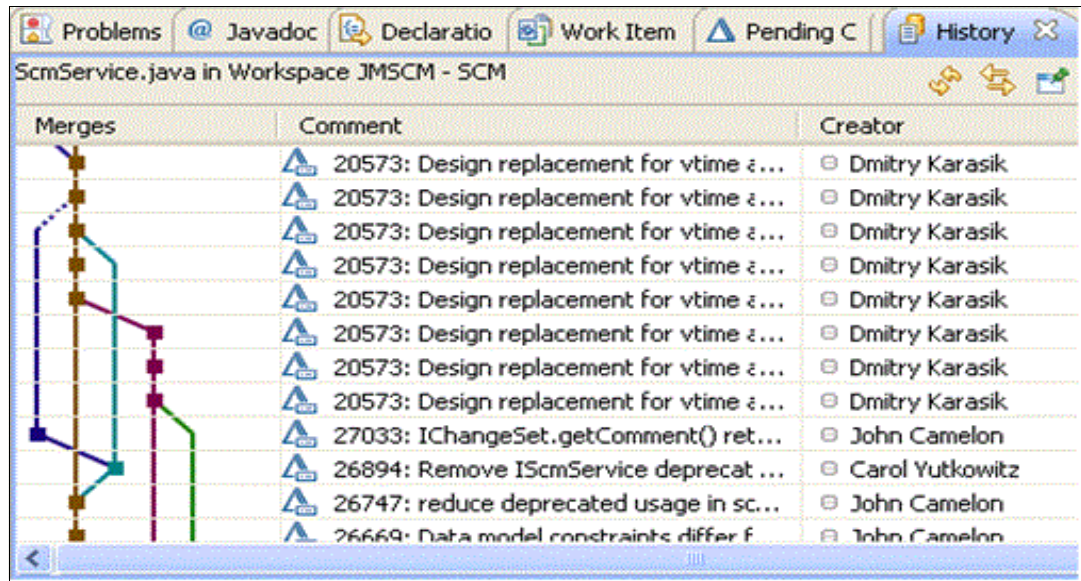


Figure 4-21 History of the changes of a source

More information

For more information about IBM Rational Team Concert, see the information center that is found at the following website:

<http://publib.boulder.ibm.com/infocenter/iadthelp/v8r5/>

The Jazz website is also a good entry point to learn more about Rational Team Concert:

<https://jazz.net/products/rational-team-concert/>

You can download the product from this site and use it at no charge during a 60-day trial period.

4.5.4 ARCAD Pack for Rational

ARCAD Pack for Rational adds more capability to both IBM Rational Developer for i and Rational Team Concert for IBM i. It is composed of four tools:

- ▶ ARCAD-Audit: Helps audit and clean applications before loading them into the change management and cross-reference repository.
- ▶ ARCAD-Observer: Provides developers with impact analysis and diagramming capabilities directly in their Rational Developer for i work area.
- ▶ ARCAD-Builder: Provides an extended function for a dependency build. It is fully embedded into IBM Rational Team Concert.
- ▶ ARCAD-Deliver: Brings a release management solution to deploy multi-levelled applications on multiple production area with a high level of traceability and security (rollback).



Interfacing

What is interfacing? For this chapter, interfacing is the ability to access data from various languages and requirements. These interfaces are the interfaces between the data and the program. These interfaces are sometimes local, that is, on the same system, and other times they are remote, that is, system to system. This chapter covers a wide range of preferred practices for accessing your native ILE data.

5.1 Stopping the confusion: Introduction to interfacing

Interfacing is a relatively small word, but one with a wide range of meanings. This section focuses on the connections between systems, objects, and languages.

The term *interface* is also used to define how individuals interact with various devices. These devices include the touchscreen interface of a smartphone or tablet, the combination of a PC's keyboard, mouse, and screen, and even the controls of a microwave oven. However, this section describes interfacing in the context of how various systems, or components of a single system, interact and pass data from one to another.

5.1.1 The past

In the past, interfacing between systems was a purely physical interaction. Exchanging data between two systems required that a magnetic tape be prepared on one system and shipped physically to another location to be read on another system at that location. Even earlier, the data transfer medium was punch cards or paper tape.

Over time, these methods gave way to various electronic communication vehicles, many of which were proprietary to the manufacturer of the computer hardware or software systems that were involved. However, no matter how the data was exchanged, it tended to be a batch-oriented process, that is, a set of data was passed from system A to system B, and the results of that processing were returned later.

Today, such processes still take place, but the data is typically in the form of XML files or Comma or Character Separated Values (CSV), and is often transferred electronically between the systems through mechanisms such as FTP.

Just as interactive terminals began to replace batch reports, over time it became necessary for the results of such processes to become available immediately. In some cases, this immediacy was achieved by having a program on one system pretend to be a terminal on the other and to retrieve the data that was returned programmatically. In fact, this is a technique that is still in use today to allow a program to enter data into existing online applications. But over time, more sophisticated and practical methods have evolved.

5.1.2 The present

Today, such interactions typically take place through a wide variety of different mechanisms and methods. Many interactions are done by using web services of one form or another, or by using stored procedure calls against a database. Generically, such processes come under the banner of service-oriented architecture (SOA).

SOA is really an evolution of modular programming practices. Modularity is described in Chapter 3, "Modern application architecture techniques" on page 29. SOA takes it one step further by building complete applications from discrete software *agents* (or *services*) that have simple, defined interfaces and are loosely coupled to perform a specific function. Such agents can be on the same system, or on two disparate systems on opposite sides of the world.

In SOA, there is a service provider and a service consumer. As you see, IBM i is positioned to play both roles.

Service providers are used by different consumers. For example, a service that validates a customer might be used from a web interface or from green screen or batch applications.

There are many other types of interfaces that are covered in this chapter. This chapter looks at the common interfaces that are used for the various languages that are supported on IBM i.

Having outlined what we mean by interfacing, it is time to look more closely at the details of the technologies and tools that are available. Let us first describe the concept of stateless interfaces.

5.2 Stateless interfaces

A stateless interface is an interface that handles a request without regard to what has happened before the current request. Modern applications generally require stateless interfaces to support event-driven, highly scalable applications. Stateless interfaces allow access to business functions to be easily rearranged and allow for a single job to handle requests for multiple sessions.

For an interface to be stateless, it must receive all information that is necessary to process a request as input parameters and must return all of the required response information. The request and response information can include key values that were obtained from earlier interface calls or that can be used in subsequent interface calls.

There are two different meanings of the word “state” that can cause confusion when discussing stateless interfaces. Stateless interfaces avoid *modal state*, in which certain operations must be done in a certain order before any other operations can be performed. However, stateless interfaces can use *accumulated state*, such as a web shopping cart, in which information is accumulated through multiple interface calls and the accumulated information is accessible through key values.

The fact that interfaces are designed to avoid modal state is one of the fundamental issues that cause problems with modernization technologies that attempt to access information through “screen scraping” techniques. Most traditional 5250 applications have some form of modal state that is deeply embedded in the programs. The programs require that screens be processed in a specific order even though other interfaces or workflows require a different order of operations. One simple example is an order entry application that requires customer information before the customer enters the order items. That order of operations is not the accepted standard for web-based applications, which generally allow the order items to be entered before the customer information. Another deep problem when using a modal state program behind a web interface occurs when a user opens multiple tabs to the application. Because the request can be made in any order from any tab, the modal state program might not be on the same “screen” as the user.

5.3 Communication

When you want application modernization, it is important that, in the communication phase, that all companies have heterogeneous platforms. You need to communicate, and you need the transactions and all information online.

By implementing messaging technologies, businesses can use a consistent approach to connectivity, and decouple the business application from the complexity of handling failures, error recovery, transaction integrity, security, and scalability.

5.3.1 IBM WebSphere MQ

You can use WebSphere MQ to enable applications to communicate at different times and in many diverse computing environments, as shown in Figure 5-1.

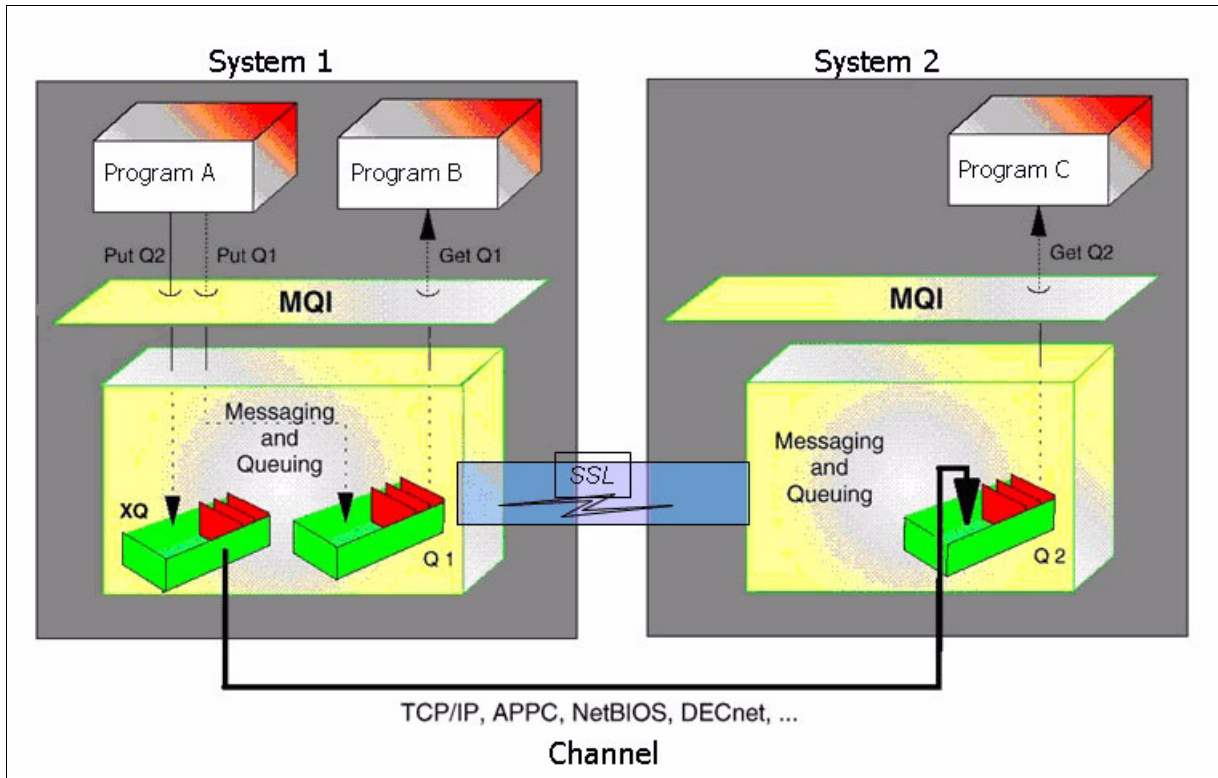


Figure 5-1 IBM WebSphere MQ flow

WebSphere MQ is messaging for applications. It sends messages across networks of diverse components. Your application connects to WebSphere MQ to send or receive a message. WebSphere MQ handles the different processors, operating systems, subsystems, and communication protocols it encounters in transferring the message. If a connection or a processor is temporarily unavailable, WebSphere MQ queues the message and forwards it when the connection is back online.

An application has a choice of which programming interfaces and programming languages it uses to connect to WebSphere MQ, as shown in Figure 5-2 on page 81.

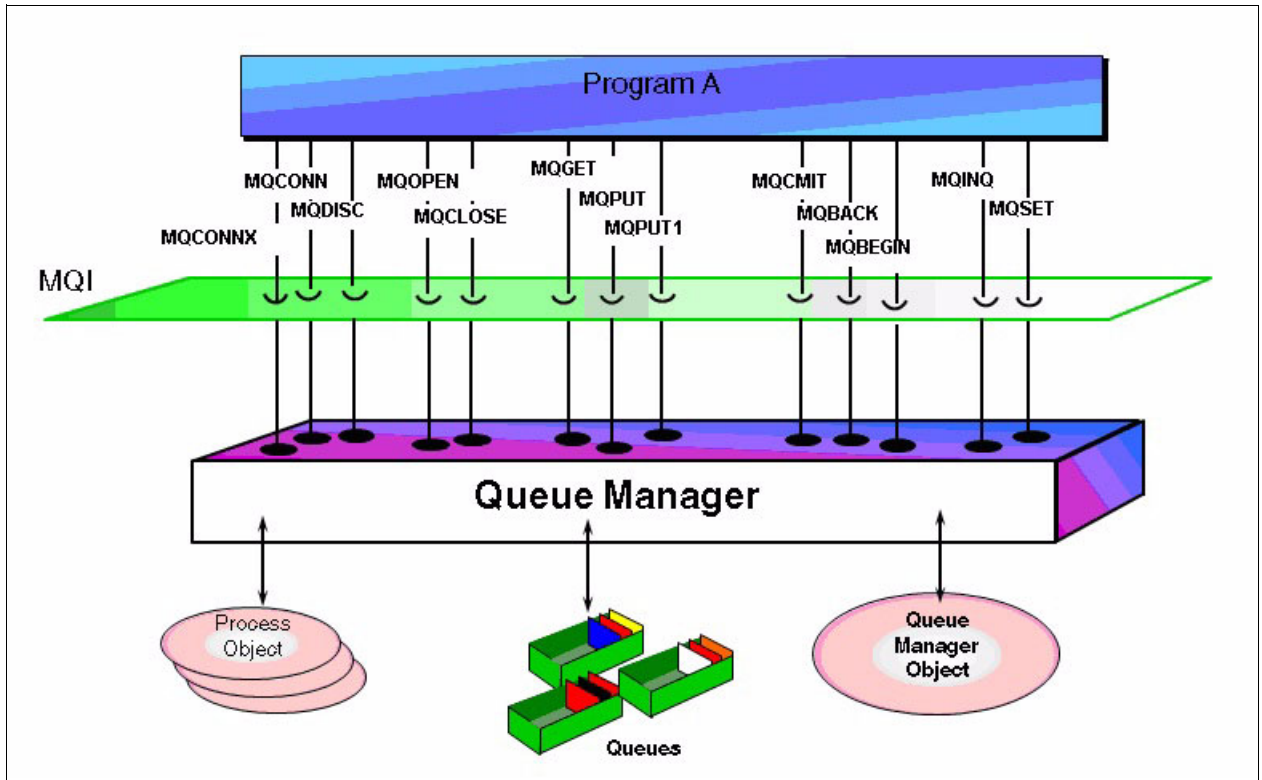


Figure 5-2 IBM WebSphere MQ interfaces

WebSphere MQ is messaging and queuing middleware, with point-to-point, publish and subscribe, and file transfer modes of operation. Applications can publish messages to many subscribers over multicast.

Messaging

Programs communicate by sending data to each other in messages instead of calling each other directly.

Queuing

Messages are placed on queues so that programs can run independently of each other, at different speeds and times, in different locations, and without having a direct connection between them.

Point-to-point

Applications send messages to a queue, or to a list of queues. The sender must know the name of the destination, but not where it is.

Publish/subscribe

Applications publish a message on a topic, such as the result of a game that is played by a team. WebSphere MQ sends copies of the message to applications that subscribe to the results topic. They receive the message with the results of games that are played by the team. The publisher does not know the names of subscribers, or where they are.

Multicast

Multicast is an efficient form of publish/subscribe messaging that scales to many subscribers. It transfers the effort of sending a copy of a publication to each subscriber from WebSphere MQ to the network. When a path for the publication is established between the publisher and subscriber, WebSphere MQ is not involved in forwarding the publication.

File transfer

Files are transferred in messages. WebSphere MQ File Transfer Edition manages the transfer of files and the administration to set up automated transfers and log the results. You can integrate the file transfer with other file transfer systems, with WebSphere MQ messaging, and the web.

Telemetry

WebSphere MQ Telemetry is messaging for devices. WebSphere MQ connects device and application messaging together. It connects the internet, applications, services, and decision makers with networks of instrumented devices. WebSphere MQ Telemetry has an efficient messaging protocol that connects many devices over a network. The messaging protocol is published so that it can be incorporated into devices. You can also develop device programs with one of the published programming interfaces for the protocol.

Scope of WebSphere MQ

The following items describe the scope of WebSphere MQ:

- ▶ WebSphere MQ sends and receives data between your applications and over networks.
- ▶ Message delivery is ensured and decoupled from the application. It is ensured because WebSphere MQ exchanges messages transactionally, and decoupled because applications do not have to check that the messages they sent are delivered safely.
- ▶ You can secure message delivery between queue managers with SSL/TLS.
- ▶ With Advanced Message Security (AMS), you can encrypt and sign messages between being sent by one application and retrieved by another.
- ▶ Application programmers do not need to have communications programming knowledge.

5.3.2 Data queues

Data queues on IBM i servers are *DTAQ type objects. You can use this kind of object for communicating different jobs on the same system or for communicating between jobs on different systems. This point is important because you do not need to build interfaces for transferring information from one system to another. A data queue is a powerful program-to-program interface.

You find this type of object only on IBM i servers. If you must communicate between IBM i and other platforms, you must use IBM WebSphere MQ.

The data queues objects have the following characteristics:

- ▶ The data queue allows for fast communication between jobs. Therefore, it is an excellent way to synchronize and pass data between jobs.
- ▶ Many jobs can simultaneously access the data queues.
- ▶ Messages on a data queue are in free format. Fields are not required as they are in database files.
- ▶ The data queue can be used for either synchronous or asynchronous processing.

- ▶ The messages on a data queue can be ordered in one the following ways:
 - Last-in first-out (LIFO): The last (newest) message that is placed on the data queue is the first message that is taken off the queue.
 - First-in first-out (FIFO): The first (oldest) message that is placed on the data queue is the first message that is taken off the queue.
 - Keyed. Each message on the data queue has a key that is associated with it. A message can be taken off the queue only by specifying the key that is associated with it.
- ▶ The operational system has all the commands for accessing to data queues; for other programming languages, there are many forms for accessing this resource.

The data queue classes support the following data queues:

- ▶ Sequential data queues: Entries on a sequential data queue on the server are removed in FIFO or LIFO sequence.
- ▶ Keyed data queues: Entries on a keyed data queue are removed according to key value.

Typical usage of data queues

Programmers who are familiar with IBM i programming are accustomed to using queues. Data queues simply represent a method that is used to pass information to another program.

Because this interface does not require communications programming, you can use it either for synchronous or for asynchronous (disconnected) processing.

You can develop host applications and PC applications by using any supported language. For example, a host application might use RPG, and a PC application might use C++. The queue is there to obtain input from one side and to pass input to the other.

The following example shows how data queues might be used, as shown in Figure 5-3.

1. A PC user might take telephone orders all day, and enter each order into a program, and the program places each request on an IBM i data queue.
2. A partner program (either a PC program or an IBM i program) monitors the data queue and pulls information from queue. This partner program can be simultaneously running, or started after peak user hours.
3. It might or might not return input to the initiating PC program, or it might place something on the queue for another PC or IBM i program.
4. Eventually, the order is filled, the customer is billed, the inventory records are updated, and information is placed on the queue for the PC application to direct a PC user to call the customer with an expected ship date and print the order.

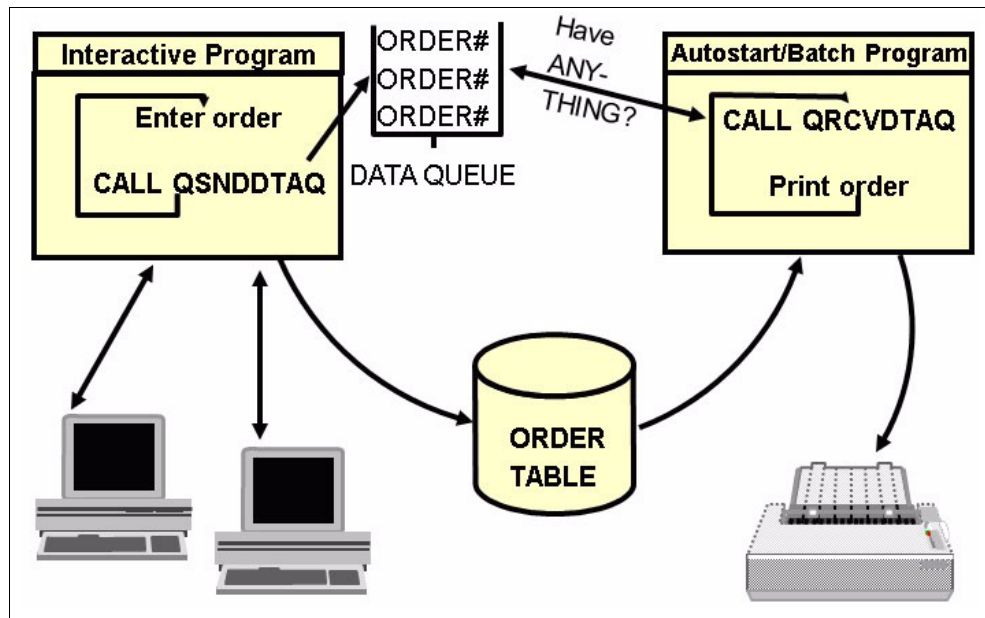


Figure 5-3 Using a data queue object to communicate with an asynchronous job

5.3.3 TCP/IP

Transmission Control Protocol over Internet Protocol (TCP/IP) is the most commonly used communication protocol in use today. It provides reliable delivery of a stream of data over interconnected networks. This reliable transport provides the basis for a wide variety of application-level protocols, including Hypertext Transfer Protocol (HTTP) for web pages and web services, Simple Mail Transfer Protocol (SMTP) for email, File Transfer Protocol (FTP) for copying files, Telnet for terminal connections, and many other standard and custom protocols.

The basic design of most TCP/IP application protocols is a server that is waiting for client connections. When a client connects to the server and establishes the reliable bidirectional TCP/IP stream, the application protocol defines the order and content of messages between the client and the server.

5.3.4 SSL/TLS

Secure Sockets Layer (SSL) was the predecessor of the current Transport Layer Security (TLS). TLS uses certificates and cryptographic protocols to secure data that is transported over TCP/IP. The standard TCP/IP applications on IBM i (HTTP, FTP, and Telnet) and the IBM i host servers can be configured to use TLS.

To use TLS for TCP/IP communications, server certificates must be generated, signed by a certificate authority (CA), and associated with the application. The CA can be the local CA that is created on the IBM i server, an internal enterprise CA, or an external public CA. The advantages and disadvantages of each of the options are described in Table 5-1.

Table 5-1 Advantages and disadvantages of CA options

Certificate authority	Advantages	Disadvantages
Local CA	Easy to generate No external steps No additional cost	Local CA certificate must be installed on all clients.
Enterprise CA	No additional cost	External steps. Enterprise CA certificate must be installed on all clients.
Public CA	Common CA certificates are preinstalled on most clients	Additional cost. External steps.

The certificates that are required for TLS are created and managed through the Digital Certificate Manager (DCM) web interface. The IBM i information center provides detailed instructions for each of the configuration steps. The specific certificates and configuration steps that are required are described in Table 5-2.

Table 5-2 Configuration steps for specific certificates

Application	Configuration
Standard application/Local CA	Create Local CA Create Server Certificate Assign Server Certificate to Application
Standard application/Enterprise CA	Create Server Certificate Request Import Enterprise CA Import Signed Server Certificate Assign Server Certificate to Application
Standard application/Public CA	Create Server Certificate Request Import Signed Server Certificate Assign Server Certificate to Application
Custom application or web server	Create Application Follow steps for Standard Application

5.3.5 HTTP/HTTPS

HTTP is the protocol that is used by web servers and web browsers. IBM HTTP Server for i (based on the open source Apache web server) can be used to serve static Hypertext Markup Language (HTML) files that are stored in the integrated file system (IFS) or in physical files, or images, and other file types (such as PDF) stored in IFS. In addition, it can serve dynamic pages in which the content is generated by programs that are written in any of the ILE programming languages. The interface between the web server and the programs follows the Common Gateway Interface (CGI) standard with some additions specific to IBM i, including the conversion between the ASCII character set that is commonly used for web pages and the EBCDIC character set that is used on IBM i.

A simple CGI program can be written in ILE RPG by creating the ILE RPG prototypes for a few C functions and using them to retrieve information about the request and send the dynamically generated reply.

The following program (Example 5-1) creates a simple web page that contains the query string (the part of the Uniform Resource Locator (URL) after the?).

Example 5-1 SimpleWeb page query string

```
h dftactgrp(*no) bnddir('QC2LE') actgrp(*new)

      * Include the C prototypes
      /copy cgipr

      * Variable to hold the query string environment data
      d envval          s          200          varying

      /free
      // Retrieve the query string (the part of the URL
      // after the ?)
      envval = %str(getenv('QUERY_STRING'));
      // The CGI interface requires that the content type be specified.
      // Content types that start with "text" will be converted to ASCII if
      // the server is configured for conversion.
      // The HTTP header is separated from the HTML content with a blank line,
      // so two CRLF strings are appended to the line.
      printf('Content-Type: text/html' + CRLF + CRLF);
      // Start the HTML output
      printf('<html><head><title>Simple Web</title></head>');
      // Complete the HTML output which includes the request query string
      printf('<body><h1>Hello, %s</h1></body></html>' +CRLF:envval);
      // The generated HTML will be sent to the browser.
      return;
      /end-free
```

The following prototype include (Example 5-2) contains the functions that are needed for the program in Example 5-1.

Example 5-2 CGIPR - function prototype include

```
*****
      * Simple prototype for C printf function that handles from
      * zero to five string substitution values
      *****
```

```

d printf          pr          10i 0 extproc('printf')
d fstring         *          * value options(*string)
d str1            *          * value options(*nopass:*string)
d str2            *          * value options(*nopass:*string)
d str3            *          * value options(*nopass:*string)
d str4            *          * value options(*nopass:*string)
d str5            *          * value options(*nopass:*string)

*****
* Prototype for C function to retrieve an environment variable
*****
d getenv          pr          * extproc('getenv')
d varname         *          * value options(*string)

*****
* EBCDIC Carriage return/line feed that will convert to ASCII properly
*****
d CRLF            c          x'0D25'
```

Although a program that produces simple HTML can hardcode the output, this process becomes cumbersome for more complex pages. One no-charge tool that can be used to map field data into HTML templates is CGIDEV2.

An HTTP server is configured by using the IBM Web Administration for i web interface. That interface provides the tools and wizards that are necessary to set up quickly a web server on IBM i. The configuration of a secure HTTP server (HTTPS) also requires the use of the DCM tool.

5.3.6 FTP/FTPS

The FTP client and server applications on IBM i provide a standard way of transferring files between heterogeneous systems. The IBM i implementations of this protocol include the capability of converting data from standard IBM i physical files (EBCDIC with fixed-length records) to and from the ASCII variable length record stream files common on other platforms. Although the FTP client is often used as an interactive application, it can be used in scripted or batch applications by creating input and output physical files and overriding the FTP input and output to those files. The instructions for doing this are available at the IBM i information center.

FTPS is an extension to FTP that uses TLS to provide secure transmission of the data. FTPS is enabled on the server by assigning a certificate in DCM and using the **CHGFTPA** command to enable the option. FTPS support on the client is specified by the **SECCNN** parameter.

5.3.7 SSH/SFTP

Secure Shell (SSH) is a protocol for providing secure communication between systems. It can be used for remote sign-in, remote command execution, file transfers, and for providing secure TCP/IP socket tunnels. The SSH File Transfer Protocol (SFTP) is often used for secure file transfer between heterogeneous machines.

The SSH protocol does not use digital certificates for authenticating keys. Instead, it uses manual verification on the first connection and stores this confirmation for subsequent connections. Another significant difference between FTPS and SFTP is that FTPS provides various automatic data conversion options, and SFTP simply transfers the data without conversion. If SFTP is used for transferring data from IBM i physical files, it is necessary to copy the data to and from IFS with the `CPYTOSTMF` or `CPYFRMSTMF` commands with the appropriate conversion options.

5.3.8 Sockets

Although there are various application-level protocols for communicating over Internet Protocol networks, there is often a need for data communication that does not match one of the standard protocols because of either application or performance requirements. The TCP/IP socket interface can be used to write custom interfaces to standard protocols, to write interfaces to protocols that are not supported by standard IBM i applications, or to write interfaces to custom protocols.

One important property of a TCP/IP socket connection is that it is a stream connection. This means that while the bytes are received in the same order that they are sent, there is no guarantee that they are received in the same blocks as they were sent. For example, if 20,000 bytes are sent in a single operation, they are not necessarily received in a single operation even if the receive function is provided with a large enough buffer. The application-level protocol must provide some mechanism to delimit messages or records that are sent over the stream. The most common mechanisms are providing a length before the message data or delimiting the messages with a delimiter character. It is easier to program the receive function if the length option is used because the message data does not have to be examined for the delimiter character.

The example code in Example 5-3 calls a method to receive a four-byte length field from a socket.

Example 5-3 Calling a method to receive a four-byte field that is returned on a socket

```

d socket          s          10i 0
d length         s          10i 0
d buffer         s          1000
d rc             s          10i 0
/free
rc = FillReceive(socket:%addr(length):%size(length));
if rc = 0;
    rc = FillReceive(socket:%addr(buffer):length);
endif;
...

```

The following code (Example 5-5 on page 90) is the code that receives a four-byte value from a socket request.

Example 5-4 FillReceive - receiving a four-byte value from a socket

```

*****
* Receive a specific number of bytes from a socket
*****
p FillReceive    b          export
*****
d FillReceive    pi          10i 0
d socket         10i 0 value

```

```

d  buffer          *  value
d  size            10i 0 value
d*
d  length          s      10i 0
/free
  dow size > 0;
    length = recv(socket:buffer:size:0);
    if length < 0;
      return length;
    endif;
    buffer += length;
    size -= length;
  enddo;
  return 0;
/end-free
p FillReceive     e

```

Server applications

Generally, a TCP/IP socket server application listens on a specific port (which is identified by a number up to 65535) for a client to connect. When a client connects, the server accepts the connection, receives the client requests, and sends the appropriate responses. It is often the case that the server must be able to handle multiple clients simultaneously. In those cases, it is necessary for the server to use multiple processes or threads. A technique that is commonly used in example code for other platforms is to create a process for each incoming client request. This does not scale well on IBM i because of the amount of processing that is necessary to start a job. A better technique is to start multiple jobs to process client requests and distribute the incoming client requests to those jobs. There are various techniques to do this task. One simple and reliable technique takes advantage of user queues to allow a dispatcher job to find an available processing job and give a client connection to that job.

Dispatcher job flow

The following list provides an outline of the process for a dispatcher job flow:

1. Create a LIFO user queue.
2. Submit a pool of processing jobs.
3. Open a listening socket.
4. Do the following until server shutdown request occurs:
 - a. Wait on **accept ()** for an incoming connection
 - i. Receive a process job identifier from a user queue.
 - ii. If none is immediately available, submit another processing job and wait on a queue for a job identifier.
 - iii. Use the **give descriptor ()** function to send a connection to a processing job.
 - b. Notify the processing jobs of the server shutdown.

Processing job flow

The following list provides an outline of the process for a processing job flow:

1. Do preconnection work (open files and other actions).
2. Do the following until a server shutdown request occurs:
 - a. Send a job identifier on a user queue.
 - b. Wait on the **take descriptor ()** for a connection.
 - c. Do the following until a protocol defined close occurs:
 - i. Receive a message from a client.

- ii. Process a request.
- iii. Send a response to a client.
- d. Close the connection.

The use of a LIFO user queue causes the most recently used job to be used for a new connection, which increases the chance that it is still active in memory. The dispatcher job can also limit the number of processing jobs. In that case, the dispatcher just holds the client connection until one of the processing jobs becomes available.

Client applications

In general, client applications are simpler than server applications because they do not have to handle multiple simultaneous connections. The client application opens a socket connection to a server, sends and receives the application defined messages, and then closes the connection.

SSL/TLS programming

There are two sets of APIs that can be used to write SSL/TLS sockets programs: an older set that start with SSL_ and the more current GSKit set. The GSKit APIs are recommended for new applications.

Useful socket declarations and prototypes

The following example (Example 5-5) contains a number of coding examples where various socket declarations and prototypes are defined.

Example 5-5 Useful socket declarations and prototypes

```

*
  * Constant values
  *
d AF_INET          c          2
d AF_INET6        c          24
d SOCK_STREAM     c           1
d SO_KEEPALIVE    c          25
d SOL_SOCKET      c          -1
d TCP_NODELAY     c           10
d IPPROTO_TCP     c           6
d SOCKOPT_SIZE    c           4

*
  * Error values
  *
d EADDRNOTAVAIL   c          3421
d EIOERROR        c          3101

*
  * Structures
  *
d sockaddr_in     ds          qualified template
d sin_family      5u 0
d sin_port        5u 0
d sin_addr        10u 0
d sin_zero        8

```



```

d hostent      ds          qualified template
d h_name      *
d h_aliases   *
d h_addrtype  10i 0
d h_length    10i 0
d h_addr_list *

d h_addr_list ds          qualified template
d address     *

d timeval     ds          qualified template
d tv_sec      10i 0
d tv_usec     10i 0

d fdset       ds          qualified template
d fdes        10u 0 dim(7)

*
* Prototypes
*

d socket      pr          10i 0 extproc('socket')
d addr_family 10i 0 value
d type        10i 0 value
d protocol    10i 0 value

d inet_addr   pr          10u 0 extproc('inet_addr')
d addr_string *          value

d setsockopt  pr          10i 0 extproc('setsockopt')
d sock_desc   10i 0 value
d level       10i 0 value
d option_name 10i 0 value
d option_value 10i 0 const
d option_length 10i 0 value

d gethostbyname pr        *   extproc('gethostbyname')
d host_name     *          value

d connect      pr          10i 0 extproc('connect')
d sock_desc    10i 0 value
d dest_addr    *          value
d addr_len     10i 0 value

d select       pr          10i 0 extproc('select')
d max_desc     10i 0 value
d read_set     likeds(fdset) options(*omit)
d write_set    likeds(fdset) options(*omit)
d except_set   likeds(fdset) options(*omit)
d wait_time    likeds(timeval) options(*omit)

d send         pr          10i 0 extproc('send')
d sock_desc    10i 0 value
d buffer       *          value

```

d	buffer_length		10i	0	value
d	flags		10i	0	value
d	recv	pr	10i	0	extproc('recv')
d	sock_desc		10i	0	value
d	buffer		*		value
d	buffer_length		10i	0	value
d	flags		10i	0	value
d	sclose	pr	10i	0	extproc('close')
d	sock_desc		10i	0	value
d	geterrno	pr	*		extproc('__errno')

5.4 SOA and web services

SOA is an IT architectural style that supports the transformation of your business into a set of linked services or repeatable business tasks that can be accessed over a network when they are needed. This network might be a local network, it might be the internet, or it might be geographically and technologically diverse, combining services in New York, London, and Hong Kong S.A.R. of the PRC as though they were all installed on your local desktop. These services can coalesce to accomplish a specific business task, enabling your business to quickly adapt to changing conditions and requirements.

When SOA implementation is guided by strategic business goals, you ensure the positive transformation of your business and can realize the chief benefits of an SOA:

- ▶ Alignment of IT with the business
- ▶ Maximal reuse of IT assets
- ▶ Reduced time to market
- ▶ Reduced costs
- ▶ Reduced business risk

The most common way to transform your business into linked services is by the use of web services. Web services allow applications to communicate with each other in a platform and programming language independent manner. A group of web services interacting together in this manner defines a particular web service application in a SOA.

The software industry is finally coming to terms with the fact that integrating software applications across multiple operating systems, programming languages, and hardware platforms is not something that can be solved by any one particular proprietary environment. Traditionally, the problem has been one of tight-coupling, where one application that calls a remote network is tied strongly to it by the function call it makes and the parameters it requests. In most systems before web services, this is a fixed interface with little flexibility or adaptability to changing environments or needs.

Web services use technologies such as Extensible Markup Language (XML) or JavaScript Object Notation (JSON) to describe any and all data in a truly platform-independent manner for exchange across systems, thus moving towards loosely coupled applications. Furthermore, web services can function on a more abstract level that can reevaluate, modify, or handle data types dynamically on demand. On a technical level, web services can handle data much easier and allow software to communicate more freely.

On a higher conceptual level, you can look at web services as units of work, each handling a specific functional task. One step above this, the tasks can be combined into business-oriented tasks to handle particular business operational tasks, and this in turn allows non-technical people to think of applications that can handle business issues together in a workflow of web services components. Thus, after the web services are designed and built by technical people, business process architects can aggregate them into solving business-level problems. To borrow a car engine analogy, a business process architect can think of putting together a whole car engine with the car frame, body, transmission, and other systems, rather than look at the many pieces within each engine. Furthermore, the dynamic platform means that the engine can work together with the transmission or parts from other car manufacturers.

What rises from this last aspect is that web services are helping to bridge the gap between business people and technologists in an organization. Web services make it easier for business people to understand technical operations. Business people can describe events and activities and technologists can associate them with appropriate services.

With universally defined interfaces and well-designed tasks, it also becomes easier to reuse these tasks and the applications that they represent. Reusability of application software means a better return on investment on software because it can produce more from the same resources. It allows business people to consider using an existing application in a new way or offering it to a partner in a new way, thus potentially increasing the business transactions between partners.

Therefore, the primary issues that web services try to tackle are the issues of data and application integration, and that of transforming technical functions into business-oriented computing tasks. These two facets allow businesses to communicate on a process or application level with their partners, while leaving dynamic room to adapt to new situations or work with different partners on demand.

Services are all about efficiency in creation, reuse for execution, and flexibility for change. Figure 5-4 shows how web services fit in to an application architecture.

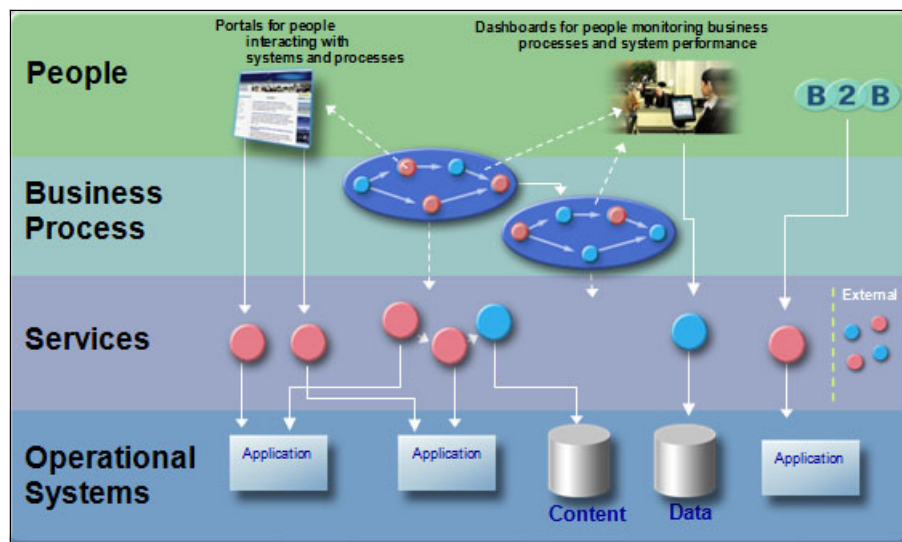


Figure 5-4 How web services fit in to an application architecture

Web services can be implemented by using SOAP or REST. The following sections describe each concept.

5.4.1 Web services that are based on SOAP

Simple Object Access Protocol (SOAP) is a messaging protocol that describes how messages should be encoded so that they can be delivered over the network by using a transport protocol such as HTTP.

Messages that are being sent are XML documents and must adhere to the protocol. As you can see in Figure 5-5, the WSDL is what defines the XML message that is being sent from the web services business function to the calling software application.



Figure 5-5 Diagram of a SOAP-based web services request

The Web Services Description Language (WSDL) is represented as a series of XML statements that constitute the definition for the interfaces of each service (that is, the contract between the client and the web service provider). The WSDL has the following rules:

- ▶ Fully describes the web service:
 - Operations
 - Parameters
 - Data types of the parameters (promotes strong data typing)
- ▶ Describes endpoints to contact the web service

Web services that are based on SOAP are generally activity-oriented, where service operations are the center of attention (resources might be involved, but serve only to refine the context within which the activity is done).

Web services that are based on SOAP are based on standards and specifications that govern various areas, such as reliable messaging, security, and transaction management.

5.4.2 Web services that are based on REST

REST is an acronym that stands for *REpresentational State Transfer*. It is an architectural style or design pattern, not a standard. The idea is that HTTP is not just the transport for a web service; it is the web service.

Messages that are exchanged between a client and a REST web service can be in any format, such as XML or JSON. As you can see in Figure 5-6 on page 95, the web service message is sent without a defined contract. The message is “self-describing”. It is the responsibility of the software application to decipher the request.

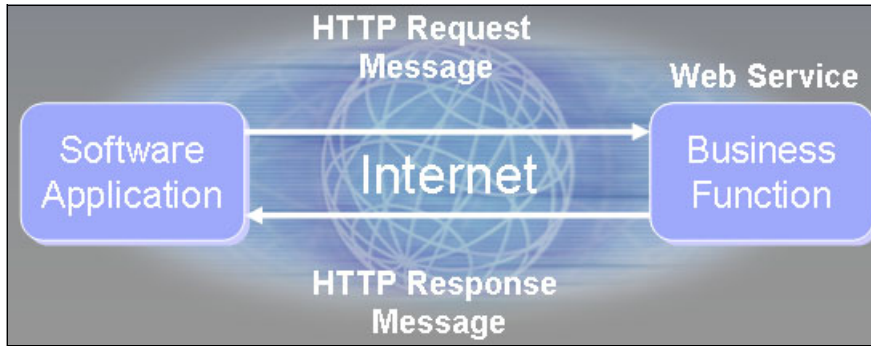


Figure 5-6 Diagram of a REST-based web services request

With REST, each resource/object that is exposed to the web has its own logical URL for accessing it. (The web server maps the URL to the object so that each object does not need its own static page or predefined URL.) HTTP protocol methods are used to denote the invocation of different operations for typical (create, retrieve, update, and delete) operations:

- ▶ POST: Create a resource.
- ▶ GET: Retrieve a resource.
- ▶ PUT: Update a resource.
- ▶ DELETE: Delete a resource.

Web services that are based on REST are resource-oriented rather than activity-oriented (that is, everything is a resource and resources are the center of attention). In addition, the web is used as the programming platform.

5.4.3 SOAP versus REST

Which style should be used? Is one better than the other? Figure 5-7 attempts to compare the two styles.

SOAP	REST
<ul style="list-style-type: none"> - Activity-oriented - Few endpoints, many custom methods - Few nouns, many verbs - Example: <code>lbmanager.getState("dave")</code> - WSDL is the contract between client and server - Lots of tooling, mainly to generate client stubs in order to communicate with Web service and handle parsing/conversion of data - Used by enterprise customers and b2b situations 	<ul style="list-style-type: none"> - Resource-oriented - Many resources, few fixed methods - Many nouns, few verbs - Example: <code>GET http://w3/lightbulbs/dave</code> - No standard, but WADL is used and may become a standard eventually - No tooling (application needs to parse the data returned and do any conversions) - Extensively used in Web 2.0 world - Lots of REST-like services, but do not necessary follow REST principles

Figure 5-7 SOAP versus REST web services

The choice of REST versus SOAP is nothing more than a choice over a design strategy that is based on business and application need, but it is a choice that can profoundly impact how an application is used and evolves over time.

Where both SOAP-based web services and REST-based web services are offered, REST-based APIs are more widely used, mainly because REST services are easily used from scripting languages, in addition to easily allowing for browser-based experimentation.

In summary, the answer is not an either/or proposition. It is perfectly okay to have both types, depending on the problem that must be solved. You might want to provide REST web services for Web 2.0 and browser-based clients, or you might want to use SOAP web services if the architecture must address complex requirements that are addressed by SOAP-based web service standards, such as transactions, security, addressing, trust, and coordination, which REST cannot provide or requires work to implement in REST.

5.4.4 Web services support on IBM i

There are various technologies for IBM i that are available in support of web services. The following sections look at some of the most popular technologies from two perspectives: the web service *provider* perspective and the web service *consumer* perspective. Figure 5-8 shows a diagram of who is the consumer and who is the provider for a web service.

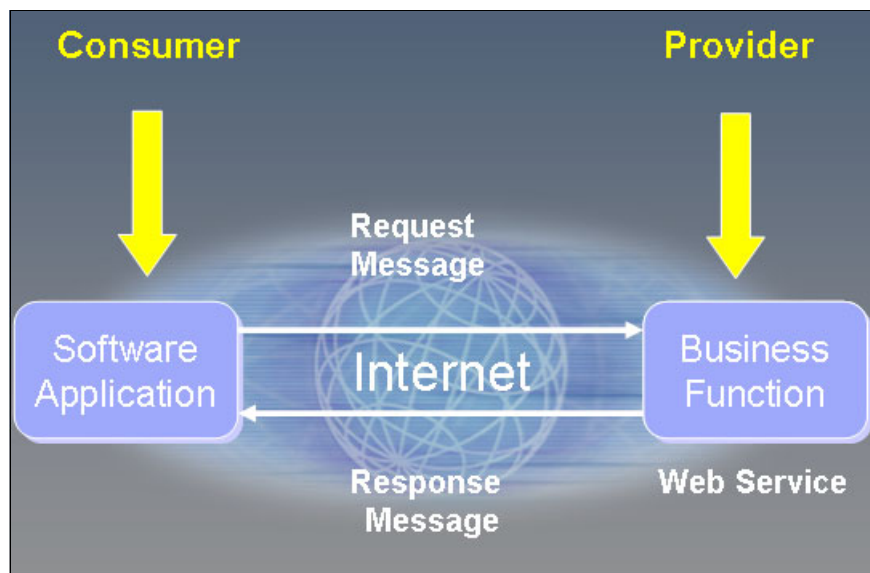


Figure 5-8 Consumer and provider of a web service request

The provider runs on the server and provides the web service. The provider might allow clients to access the web service through SOAP, REST, or both. The consumer is the client application (for example, browser or client application). The client application might use SOAP or REST to consumer the web service.

5.4.5 Web services: Provider perspective

The sections that follow describe some technologies that allow you to deploy ILE programs and service programs as web services. For each technology, there is a high-level introduction that is followed by information about whether there is support for SOAP or REST web services, in addition to where you can obtain more detailed information.

But before you get into the details of the various technologies, there is an important point to emphasize when implementing web services, and that is the characteristics of a web service. When you create a web service on IBM i, most often you are attempting to provide some business logic to an interface outside your core application. Because you are dealing with an ILE or services program, there are some important concepts about that ILE program that must be considered. Figure 5-9 compares the traditional, monolithic model of creating applications to how web services are implemented.

Web Service	Traditional Program
<ul style="list-style-type: none"> • Encapsulated <ul style="list-style-type: none"> – Access through interface • Reusable <ul style="list-style-type: none"> – Write once – use everywhere • Stateless <ul style="list-style-type: none"> – Information not retained • Event driven <ul style="list-style-type: none"> – No required order • Loosely coupled <ul style="list-style-type: none"> – Callable from anywhere 	<ul style="list-style-type: none"> • Global data <ul style="list-style-type: none"> – Access directly • Reuse by copy <ul style="list-style-type: none"> – Maintain everywhere • Stateful <ul style="list-style-type: none"> – Information retained in job • Application driven <ul style="list-style-type: none"> – Fixed order • Tightly coupled <ul style="list-style-type: none"> – Tied to application

Figure 5-9 Web services versus traditional programming methods

As you can see, it is important that you understand that web services should be coded as granularly as possible, with no dependence on state information. Web service requests can come from anywhere and, when the documented interface is followed, the implementation can change as frequently as you want.

5.4.6 Integrated web services server

The web service engine or run time is integrated in to IBM i and is used to externalize ILE business logic as a service. This integration opens the IBM i system to various web service client implementations, including RPG, COBOL, C, C++, Java, .NET, PHP, WebSphere Process Server, ESB, and Web 2.0.

Here are the features of the integrated web services server:

- ▶ Easy to use through centralized configuration and control

The web services server focuses on making the deployment of ILE-based web services as painless as possible by hiding the complexities of the web services server behind an easy to use and intuitive web administrative GUI front end that allows you to manage and monitor the server and any deployed web services.

- ▶ Leading edge

Even though the focus is on ease-of-use for the deployment of ILE-based services, the web services server is built on the powerful, yet lightweight, integrated application server and best-of-breed technologies in support of web services. The web services server supports the following items:

- Dynamic generation of WSDL documents
- WSDL 1.1, SOAP 1.1, and SOAP 1.2
- SSL and basic authentication through an HTTP server front end

- ▶ Small footprint

Uses ILE programming architecture for minimal consumption of IBM i resources.

Here are the benefits of the integrated web services server:

- ▶ Get started with no up-front cost. The integrated web services server support is part of the operating system, so you can get services up and running and show tangible results fast.
- ▶ Allows you to focus on your core capabilities. Solution providers do not require new skills or tools for the deployment of ILE-based web services and thus can focus efforts on building differentiated value for customers instead of rewriting core infrastructure.

Let us emphasize that no other product or mechanism that we are aware of enables the deployment of an ILE program object as a web service as simply as the integrated web services server. The support is integrated in IBM i. Ensure that you have the latest HTTP Server Group PTF for the release you are on and you are ready to go.

Figure 5-10 shows the flows between a web service consumer and the web service provider.

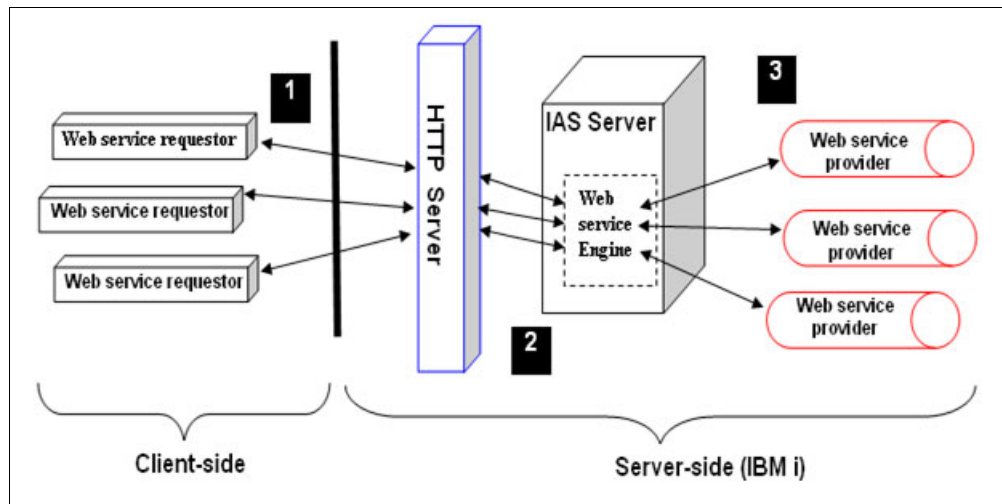


Figure 5-10 Client and server flow diagram

Here is the flow:

1. A client application starts a web service. The invocation is through a URL and the request goes through the HTTP server. The HTTP server is needed to provide SSL support and basic authentication (which is done through the Web Administration for i GUI). You can go directly to the integrated application server (IAS) instance if SSL or basic authentication is not needed.
2. The HTTP server recognizes that the request is for a web service and forwards the request on to web services engine that is running in an IAS instance.
3. The web services engine maps the request to a deployed web service, deserializes the SOAP request, and passes the data to the web service. When the web service completes by returning output parameters, the web services engine serializes the data into a SOAP response and sends the SOAP response to the HTTP server, which forwards the response back to the client.

How does the web services engine start an ILE COBOL/RPG program object (that is, program or service program)? The key to making all this work is Program Call Markup Language (PCML). PCML is based on the XML, and is a tag syntax that you use to describe the input and output parameters of programs and procedures that are contained in service programs. When you deploy a web service to an integrated web services server, the PCML is used to generate Java proxy code that performs the call to the web service implementation code that is written in RPG or COBOL.

PCML can be stored as part of the module (*MODULE) object when you compile your code by running CRTRPGMOD or CRTCLMOD. When you deploy your ILE program object as a web service, the PCML is retrieved from the program object and then the Java proxy code is generated. (Alternatively, you can specify a path to the PCML file to use if you do not want to bundle the PCML data in your module.)

Now that you have an idea of the inner workings of the integrated web services server, what are the steps to get started? The first step is to create a web services server. The second step is to deploy the ILE program object as a web service.

Creating the web services server

You can create the integrated web services server by using the IBM i Web Admin GUI Create Web Services Server wizard (see Figure 5-11). Point your browser to the Web Administration for i GUI by specifying the following URL:

http://hostname:2001/HTTPAdmin

The “host name” is the host name of your server. Sign on. (You must have *ALLOBJ authority to create a web-services server or have permission to create web services servers.) Then, start the Create Web Services Server wizard by either clicking the link in the navigation bar under the Common Tasks and Wizards heading, or on the main page of the Setup tab.

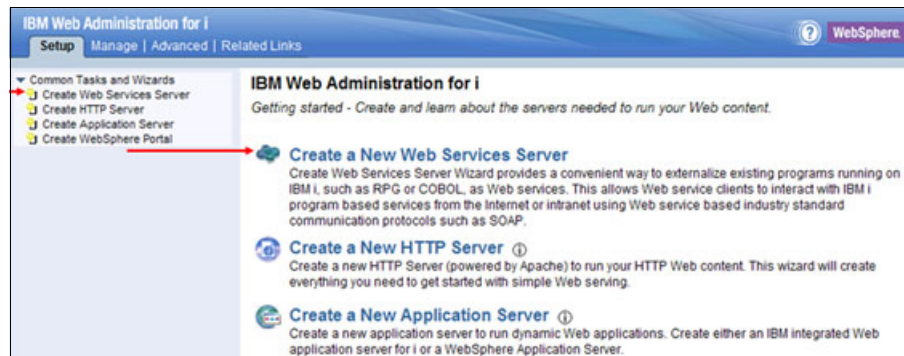


Figure 5-11 Web Admin GUI Interface

The wizard, which is shown in Figure 5-12 takes you through three windows. You are creating only the web services runtime engine. In the first window, specify the name of the server or use the default name.



Figure 5-12 Create Web Services Server wizard

The next window allows you to specify the runtime user profile for the server. You can also use the default user profile, QWSERVICE. The final window is a summary window, and when you click **Finish**, the wizard goes through the process of creating the server.

When the server is created, it starts. Figure 5-13 shows what you see when the server is created.



Figure 5-13 Manage Web Services Server in the running state

You can see the status of the server (upper left corner), in addition to the sample web service, ConvertTemp, that is deployed when the server is created.

Now, you can deploy an ILE program object as a web service.

Deploying an ILE program object as a web service

You can deploy a web service by clicking the **Deploy New Service** link, as shown by the arrow in Figure 5-13.

Clicking the **Deploy New Service** link displays a wizard that allows you to install an ILE program object as a web service. We do not show all the windows of the wizard, but we show the ones where we need to make an important point. The first window that is shown in Figure 5-14 on page 101 is the window where you specify the program object to be deployed. In this example, we deploy a service program that is called NFS400_THR in library IWSRII.

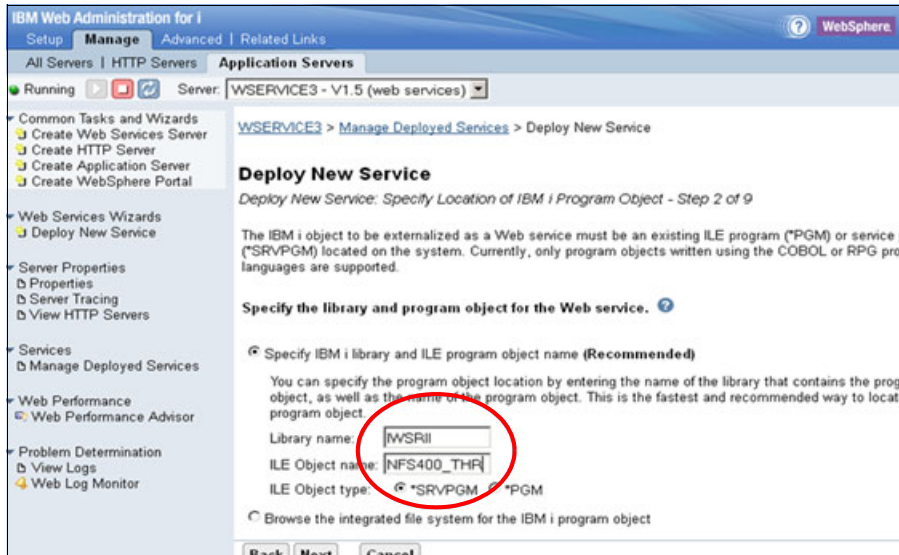


Figure 5-14 Specifying the ILE program or service program to create as a web service

The next window that is shown allows you to specify the name of the web service. The default is the name of the program object. The next window lists all the procedures in the program object that contain corresponding PCML data (see Figure 5-15). The wizard determines whether the program object has any PCML data that is stored in it. If it finds none, it shows a prompt where you can specify a path to a PCML document that describes the program object.

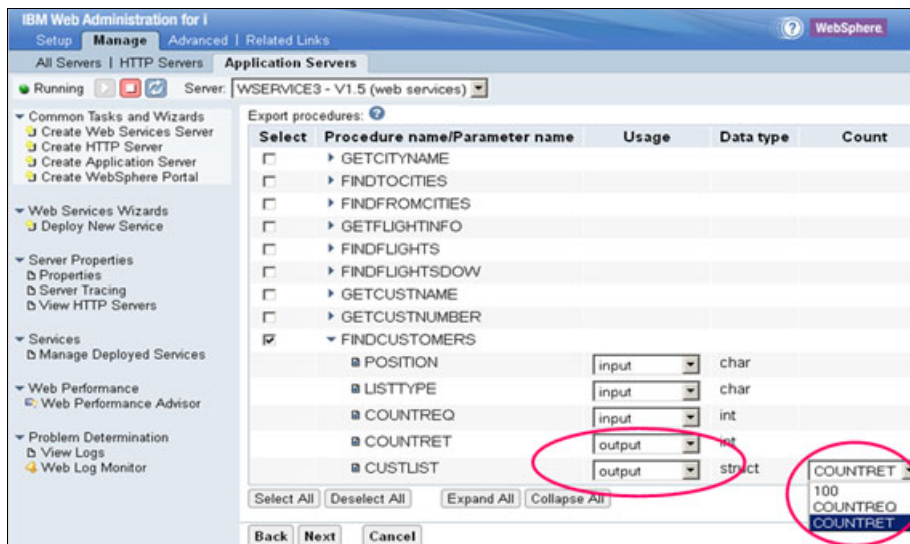


Figure 5-15 List of procedures and PCML definitions

The wizard displays a list of exported procedures. For service programs (object type *SRVPGM), there might be one or more procedures. For programs (object type *PGM), there is only one procedure, which is the main entry point to the program. In our example, we select one procedure, FINDCUSTOMERS, to be externalized as a web service operation within the service program that contains many procedures.

Expanding the procedure row shows the parameters for the procedure and various parameter attributes. The parameter attributes are modifiable. In most cases, you want to modify the parameter attributes to control what data is to be sent by web-service clients and what data is to be returned in the responses to the client requests, as shown by the circles in Figure 5-15 on page 101.

The Count column indicates that the parameter is an array. If the count value is a number, the number identifies the number of entries in the array. If the count value references another parameter, which must be an integer type, then the referenced variable is used at run time to determine the number of elements in the array. There are performance ramifications to consider regarding parameters that are arrays. If the count value is a number, and the usage of the parameter is output or input/output, then all of the array elements are returned in the response to a client request, even if the valid elements in the array are fewer than the maximum. To improve the performance of web-service requests, it is a good idea for output arrays to have the count value reference another parameter that contains the actual number of elements in the array if you know the number of elements in the array is not constant. For nested arrays, such as arrays within structures, the count value cannot be set. To change the nested array's count value, a manually constructed PCML document must be provided.

When you have selected the procedure to expose as a web service operation, the wizard shows a window where you can specify the user profile in which the web service (that is, the ILE web service implementation program object) runs. The default is to run the web service under the server user profile.

The next window (Figure 5-16) allows you to specify any libraries that must be added to the library list when running the ILE web service implementation program object. In this example, we need to add the FLGHT400 library that contains all the database files that are used by the ILE program object.

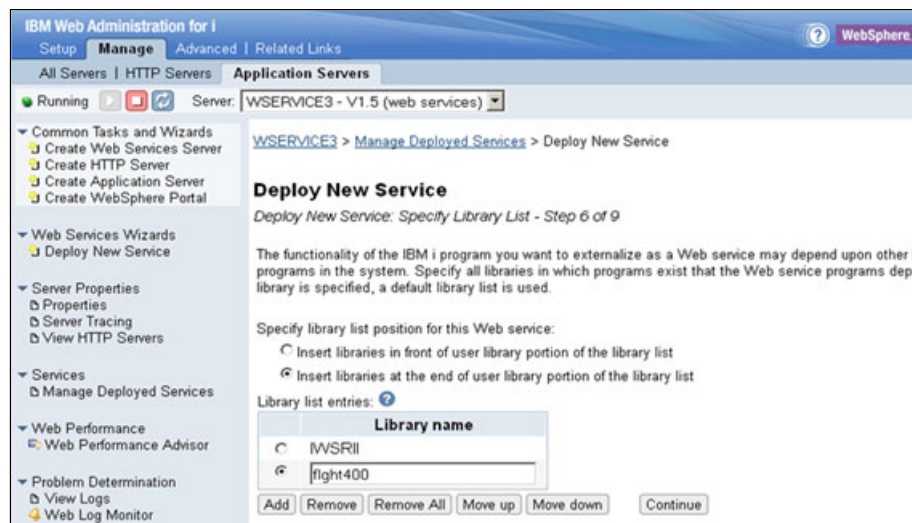


Figure 5-16 Specifying library list for web services call

The next window enables you to indicate what HTTP transport information to pass to the ILE program object (done through environment variables). The last window allows you to set the properties of the generated WSDL file.

When you click **Finish**, you see the deployed services active, as shown in Figure 5-17 on page 103.

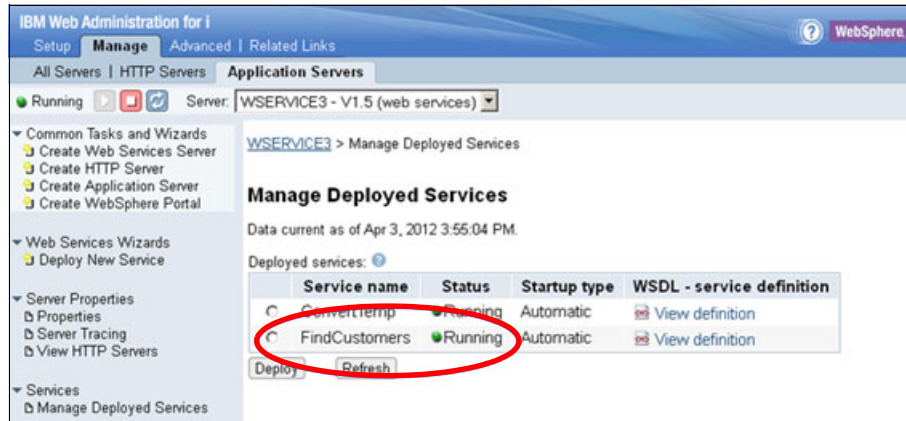


Figure 5-17 Managed web services showing the active service

The whole process takes less than five minutes. If the server is already created, deploying a web service takes less than two minutes.

Integrated web services server supports only SOAP-based web services.

For more information about integrated web services support, see the following website:

<http://www.ibm.com/systems/power/software/i/iws/>

5.4.7 WebSphere Application Server and IBM Rational Developer for i

IBM Rational Developer for i is an integrated development environment (IDE) that is built on the Eclipse platform. Designed for creating and maintaining applications on IBM i systems, it is on the developer's desktop, supporting development in both host-connected and disconnected modes.

Rational Developer for i is needed to deploy ILE program objects as web services in a WebSphere Application Server. Rational Developer for i goes through a similar process to deploy a web service to a WebSphere Application Server as the one that is used when deploying web services to the integrated web services server (for more information, see 5.4.6, "Integrated web services server" on page 97), relying on PCML to generate Java proxy code that calls the web service implementation code, an ILE program, or service program. The significant difference between the two is the structure of the package that is created to deploy the web service to the application server.

The following information about Rational Developer for i is based on IBM Rational Developer for i V9.0.

There are several editions of Rational Developer for i, depending on your needs. To deploy ILE programs and service programs as web services in a WebSphere Application Server, choose the following version: RPG and COBOL + Modernization Tools, Java Edition. As you install the package, which is done through the IBM Installation Manager, also select the option to check for updates so you get the latest fixes that are available.

You can start IBM Rational Developer for i V9.0 from the desktop environment or a command-line interface. When it is started, you see something similar to Figure 5-18.

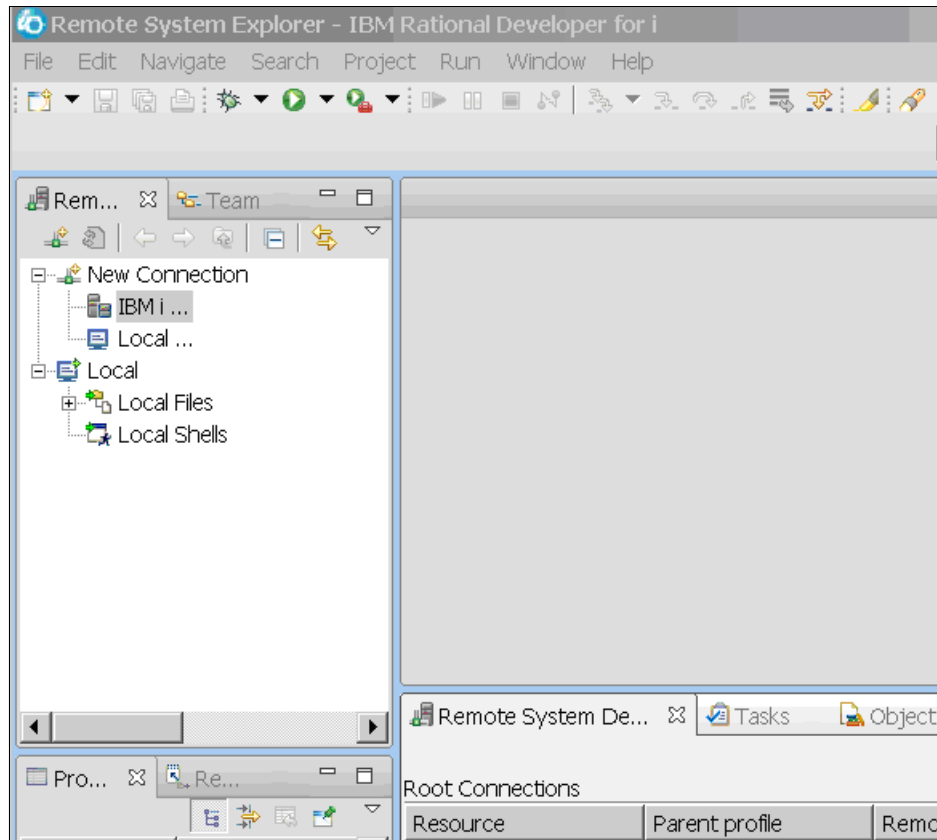


Figure 5-18 Rational Developer for i starting point

Before you begin the process of generating a web service, start the application server before running the Web Service wizard by clicking **Window** → **Show View** → **Other** → **Server** → **Servers**, right-clicking a server in the list, and select **Start**.

Deploying an ILE program object as a web service

Now, you are ready to go through the process of generating a web service. You can create a web service from ILE RPG or COBOL source or from PCML files by using the Web Service wizard. In this example, we use an RPG source file on an IBM i system that is named lp28ut24. To access the source file, create a connection to the system from Rational Developer for i. After you create a connection, you can perform actions on remote files and folders, such as editing, compiling, running, and debugging.

To configure your first connection to a remote server, complete the following steps:

1. Switch to the Remote System Explorer perspective. The name of your perspective displays at the top of the workbench. If you are not in the Remote System Explorer, then, from the workbench menu, click **Window** → **Open Perspective** → **Other** → **Remote System Explorer** → **OK**.
2. In the Remote Systems view, New Connection is automatically expanded to show the various remote systems types that you can connect to through the Remote System Explorer. Double-clicking 'IBM i ...' brings up a prompt.
3. In the Host Name field, enter the name or TCP/IP address of your remote server. In this example, we enter lp28ut24.

4. Click **Finish** to create the connection.

The Rational Developer for i perspective connects to the server and you see the connection, as shown in Figure 5-19.

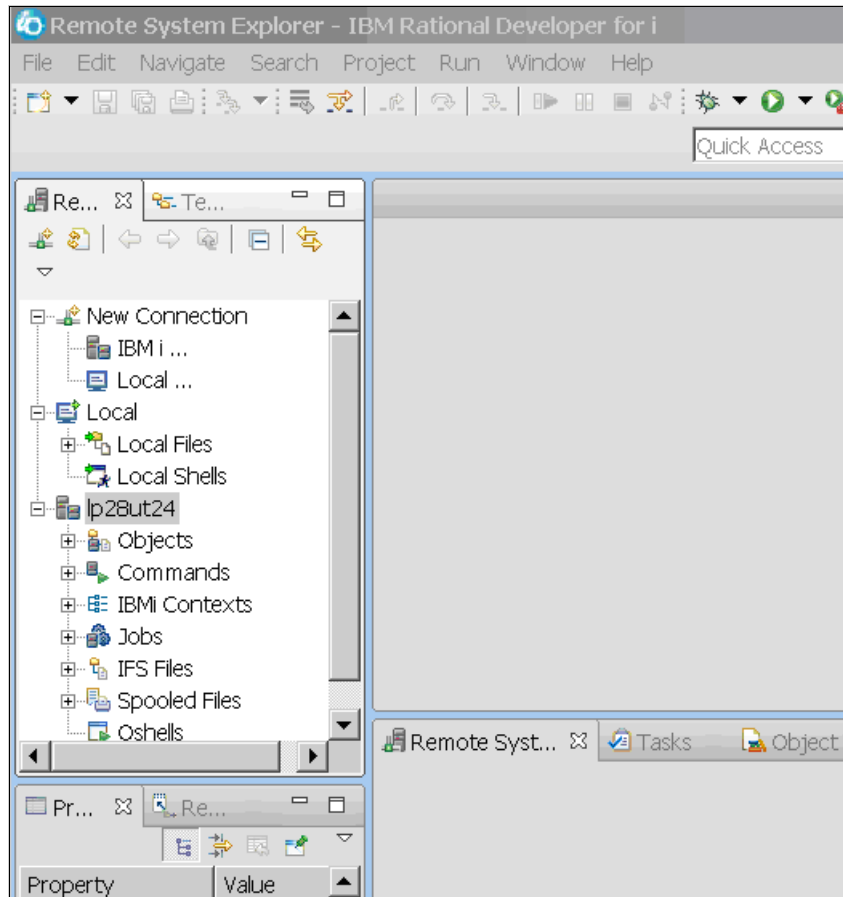


Figure 5-19 Rational Developer for i with the remote systems set

Next, you must go to the RPG source file that contains the back-end code for the web service implementation code. Before you can create the web service, you must add any libraries that the code needs to compile successfully. This is necessary in order for Rational Developer for i to generate the PCML data it needs to create the web service. In this example, the library that is needed is FLGHT400, which contains the database files, and AWSSAMPLE, which contains the include file FINDCUSTPR. To add to the library list, you expand **Objects**, right-click **Library list**, select **Add Library List Entry**, enter the name of the library in the Additional library field, and click **OK**. You then do the same operation again to add the AWSSAMPLE library.

Figure 5-20 shows the prompt where you add the library to the library list.

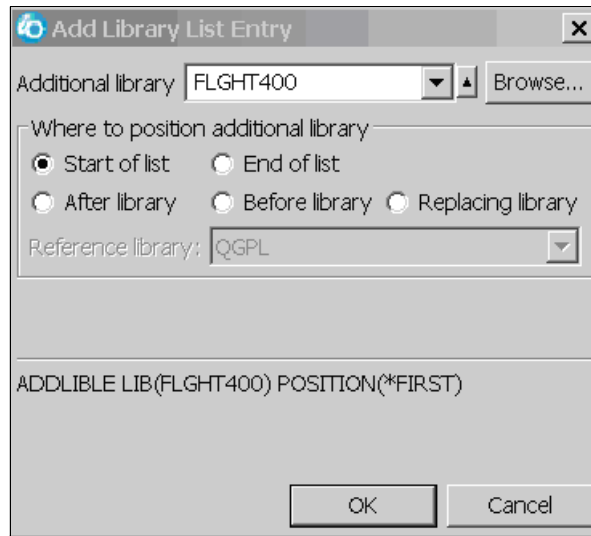


Figure 5-20 Set Library List for ILE program

The web service implementation code is in library AWSSAMPLE, file QRPGLSRC, member FINDCUST. The FindCustomers procedure searches a database for customers and returns that requested number of customers that match. Expand the user library all the way to the file, and click the source to bring the file in to Rational Developer for i, as shown in Figure 5-21.

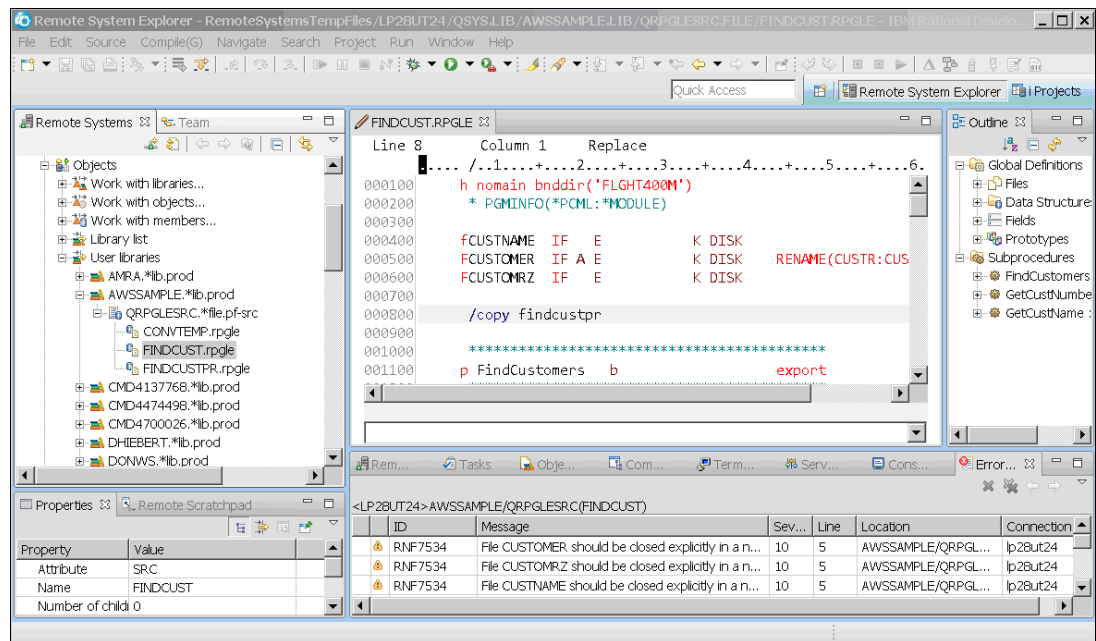


Figure 5-21 View RPG Source in Rational Developer for i

Right-click the ILE RPG source object and select **Web Services** → **Create Web Service**. The Web Service wizard opens and is populated with data from the source object that you selected, as shown in Figure 5-22 on page 107.

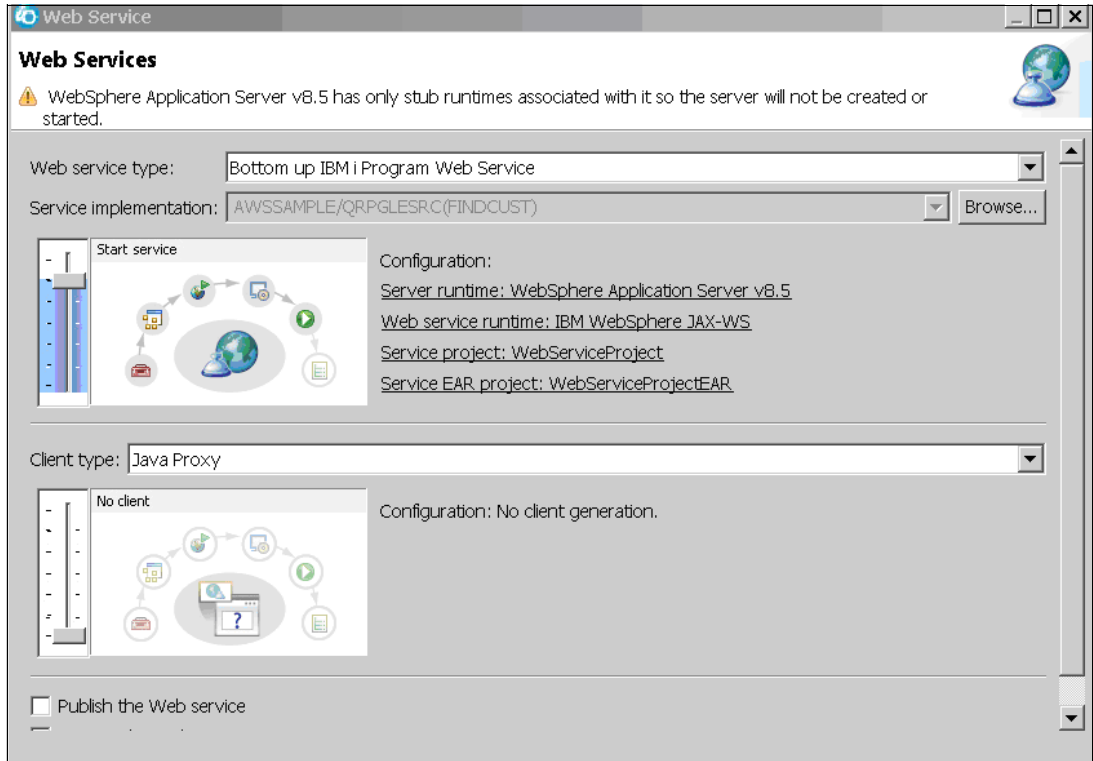


Figure 5-22 Rational Developer for i Web Services Create wizard

In the Web service type field, Bottom up IBM i Program Web Service, should be highlighted. If not, select it from the drop-down list.

Click **Browse** to open the Select Service Implementation window, as shown in Figure 5-23.

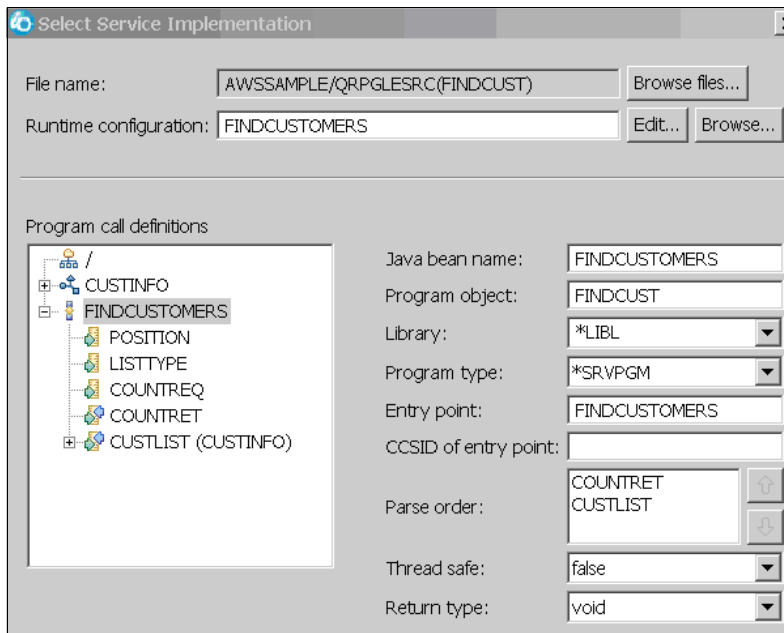


Figure 5-23 Create Web Services wizard - select Service Implementation step

The web service implementation code in this example is compiled as a service program. Ensure that the Program type is correctly set to *SRVPGM. As you can see in Figure 5-23 on page 107, the FINDCUSTOMERS procedure has five parameters: POSITION, LISTTYPE, COUNTREQ, COUNTRET, and CUSTLIST. POSITION, LISTTYPE, and COUNTREQ are input parameters; COUNTRET and CUSTLIST are output parameters. Ensure that the parameter usage is set correctly by selecting each of the parameters and explicitly setting them as either “input”, “output” or “input & output” as appropriate.

Clicking POSITION shows the window that is shown in Figure 5-24.

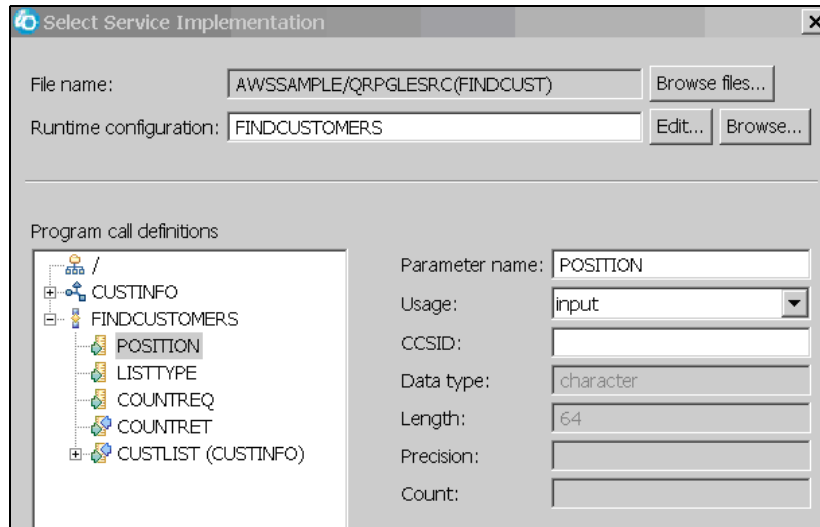


Figure 5-24 Updating the parameter usage for the POSITION field

In this case, the usage is set correctly; the parameter is an input parameter. However, if you click the COUNTRET parameter, the usage is set to “input & output”, which is not correct. Update the usage by clicking the drop-down list to set the parameter as “output”, as shown in Figure 5-25.

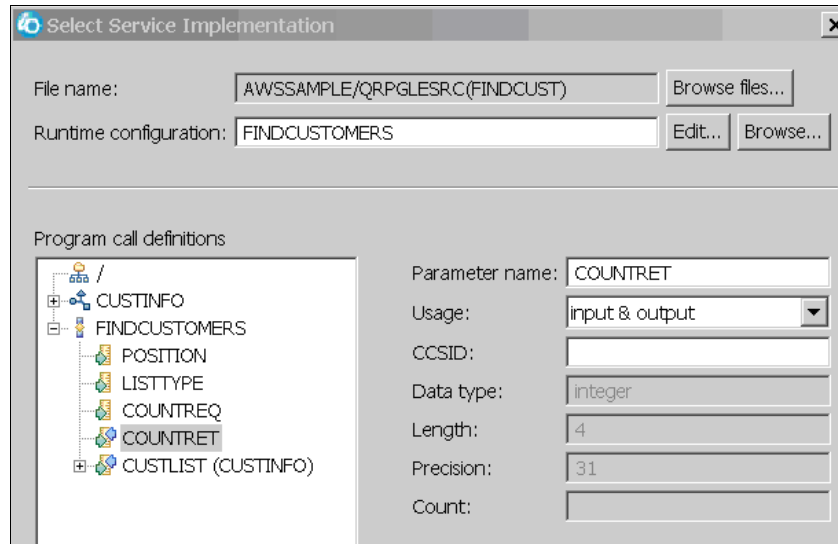


Figure 5-25 Updating the parameter usage to OUTPUT for the COUNTRET field

Next, check an update to the CUSTLIST parameter. Click the CUSTLIST parameter to update the usage. Update the usage from input & output to output. This parameter is an array of type CUSTINFO, and the count attribute for this field is preset to the maximum size of the array as defined in the RPG source, as shown in Figure 5-26.

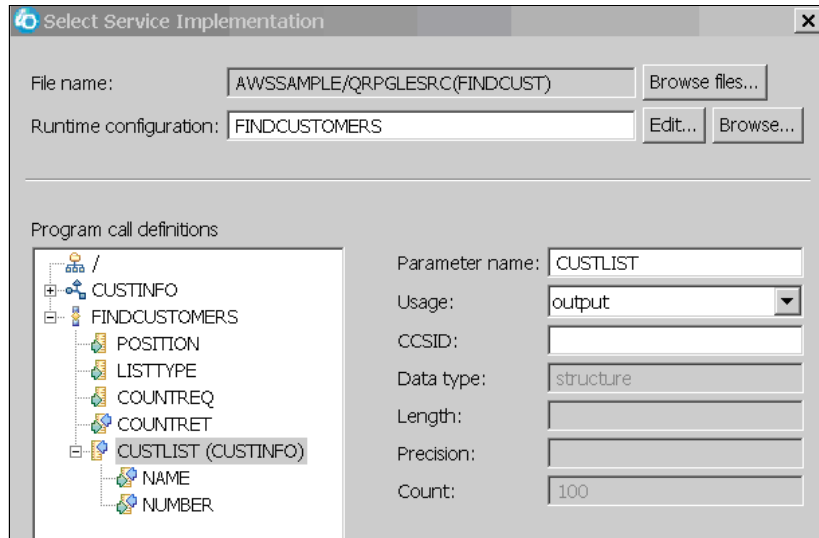


Figure 5-26 Updating the parameter usage for CUSTLIST

Here is an important point: You do not want to return all 100 records from the generated web service if only the first 10 records contain real data. You want the web service to return only the 10 records. This reduces the time and processing that is required to generate and send the response SOAP message from the web service back to the caller. We do not show this in this example, but you can generate a PCML file for the web service implementation code and change the count value so that it references the COUNTRET variable. Then, you use that PCML file to generate the web service. This ensures that the number of array elements that are returned is the actual number and not the maximum size of the array.

Now, we must set runtime configuration values. Click **Edit** next to the Runtime configuration field to set authentication and runtime configuration values. In addition to setting user ID and password information, set libraries that the program needs. In this example, the libraries that we add are FLGHT400 and AWSSAMPLE.

After setting the values, click **OK** and to return to the first page of the Web Service wizard.

Select the stages of the web services development that you want to complete by using the slider:

- ▶ **Develop:** This stage develops the WSDL definition and implementation of the web service. It includes such tasks as creating modules, which contain the generated code, WSDL files, deployment descriptors, and Java files, when appropriate.
- ▶ **Assemble:** This stage ensures that the project that hosts the web service or client is associated to an EAR file when required by the target application server.
- ▶ **Deploy:** This stage creates the deployment code for the service.
- ▶ **Install:** This stage installs and configures the web module and EAR files on the target server.

- ▶ **Start:** This stage starts the server after the service is installed on it. The server-config-wsdd file is generated.
- ▶ **Test:** This stage provides various options for testing the service, such as using the Web Service Explorer or sample JSP files.

In this example, we take the default and start the service. The Server option displays the default server. To deploy your service to a different server, click **Server** and specify the server that you want to use. The web service run time displays the default run time. To deploy your service to a different run time, click the web service run time and specify the run time that you want to use. Finally, ensure that the **Publish the Web service** check box is checked if you want to deploy the web service to the application server.

Either click **Finish** to create the web service or click **Next** to configure advanced options. In this example, we click **Finish**.

For every program that is defined in the Web Service wizard, two web service operations are generated. You can use either one of the operations to get the data that is returned from the remote program call.

The first operation returns the output data in XML format. The web services client can retrieve the data by parsing the XML string. The second operation returns the output data in the YourBeanNameResult.java class. The web services client can retrieve data by using the getter methods from the class.

Here are the advantages of using the Rational Developer for i Web Service support:

- ▶ You can deploy to a WebSphere Application Server. Deploying to a full WebSphere Application Server server gives you several advantages:
 - High availability and clustering support through a Network Deployment version of the WebSphere Application Server
 - Access to advanced Security support through WS-Security support
- ▶ You have access to the Java code that is generated. You can customize the Java code that is generated.
- ▶ You can run the web application server on a different system than your back-end ILE program.

For more information about the web services support that is part of Rational Developer for i along with an interactive sandbox where you can try these predefined exercises, see the Rational Developer for i location on IBM developerWorks:

http://public.dhe.ibm.com/software/dw/rational/emi/Building_a_web_service_from_an_RPG_program_using_Rational_Developer_for_Power_Systems_Software.pdf

5.4.8 Enterprise service bus

IBM Integration Bus Advanced, formerly known as WebSphere Message Broker, is an enterprise service bus (ESB) that provides connectivity and universal data transformation for SOA and non-SOA environments. Now, businesses of any size can eliminate point-to-point connections and batch processing regardless of platform, protocol, or data format.

You can accomplish the following tasks with IBM Integration Bus Advanced:

- ▶ Use robust capabilities to address diverse integration requirements to meet the needs of any size project.
- ▶ Help your entire organization make smarter business decisions by providing rapid access, visibility, and control over data as it flows through your business applications and systems.
- ▶ Connect throughout an array of heterogeneous applications and web services, removing the need for complex point-to-point connectivity.
- ▶ Provide extensive support for Microsoft applications and services to make the most of your existing Microsoft .NET skills and software investment.
- ▶ Deliver a standardized, simplified, and flexible integration foundation to help you more quickly and easily support business needs and scale with business growth.

What you can do with an ESB is have the ESB receive client requests, in any format, and then have the ESB transform the request and send it on to the ILE web service implementation code through another transportation protocol, such as through message queues.

The IBM ESB supports various protocols, including SOAP and REST.

More information about Integration Bus Advanced can be found at the following website:

<http://www.ibm.com/software/products/en/integration-bus-advanced>

5.4.9 Integrated web services client for ILE

The web services client for ILE delivers a mechanism to generate service artifacts and allow ILE (RPG, COBOL, C, and C++) to act as a services consumer. It can call various web service implementations, including RPG, COBOL, C, C++, Java, PHP, .NET, WebSphere Process Server, ESB, and so on.

Here are the features of the integrated web services client for ILE:

- ▶ Uses quick interoperability of ILE programming architecture for minimal consumption of IBM i resources with a small system footprint
- ▶ Supports document/literal style only
- ▶ Complies with Web Services Invocation (WSI) 1.1 basic profile
- ▶ Supports SSL

The benefits of the integrated web services client for ILE include:

- ▶ No up-front costs
- ▶ Uses and enhances existing IBM i development skills to interact with web services and SOA

Here is how web services client for ILE works. SOAP-based web services are based on files that are called WSDLs, which are XML files that contain all the information that relates to services that are available at a particular location on the internet. At their simplest level, WSDLs describe request and response message pairs in detail, and contain all the information that is relevant to the web service. The web services client for the ILE package provides a shell script, `wsd12ws.sh`. This tool enables you to turn a WSDL into a suite of RPG, C, or C++ stubs and data objects that you can call and pass information to perform a web service operation. The stubs hide the internet communication and data transformation from the application writer.

Let us go through an example of creating an RPG application that uses RPG stubs to start a web service that converts a temperature from Fahrenheit to Celsius. Here are the steps:

1. Generate the RPG stub.
2. Create an application that uses the stub.

Generating the RPG stub

In our example, we create the stubs and the service program all in one step by using the Qshell command that is shown in Figure 5-27.

```
wsdl2ws.sh -o/convtemp/RPG -lrpg  
-s/qsys.lib/cvttemp.lib/wsrpg.srvpgm  
http://lp28ut24:10022/web/services/ConvertTemp?wsdl
```

Figure 5-27 Creating the RPG Stub program based on the WSDL document

If you examine the command, you see that we are indicating to the `wsdl2ws.sh` tool that RPG stub code should be generated by using the `-lrpg` option and stored in `/convtemp/RPG`, and that a service program, `/qsys.lib/cvttemp.lib/wsrpg.srvpgm`, should be created by using the generated stub code. Also, the WSDL file is obtained from the web service by specifying the URL with a suffix of `?wsdl`. Alternatively, you can specify a file that is stored on the IBM i system.

Consider the following items:

1. RPG stubs are built on top of the C stubs, so when generating stubs, you see C and RPG files.
2. Compile failures can occur because of the 16 M (IBM i 5.4 has 65 K) limit of data structure size and individual fields. The `wsdl2ws.sh` shell scripts contain parameters for what string sizes (`-ms`) and array sizes (`-ma`) should be, so you might need to adjust those parameters to get under the limit.

The output that is generated by the `wsdl2ws.sh` tool is shown in Figure 5-28. In addition to the RPG files, C files are also generated because the RPG stub code is built on top of the C stub code. For all practical purposes, the C stub code can be ignored.

```
ConvertTempPortType_util.rpgle  
ConvertTempPortType_util.rpgleinc  
ConvertTempPortType_xsdtypes.rpgleinc  
ConvertTempPortType.c  
ConvertTempPortType.cl  
ConvertTempPortType.h  
ConvertTempPortType.rpgle  
ConvertTempPortType.rpgleinc  
CONVERTTEMPInput.c  
CONVERTTEMPInput.h  
CONVERTTEMPResult.c  
CONVERTTEMPResult.h
```

Figure 5-28 `wsdl2ws.sh` tool generated output

Here is a description of each RPG file that is generated:

- ▶ ConvertTempPortType_util.rpgle: RPG utility routines
- ▶ ConvertTempPortType_util.rpgleinc: RPG utility routines include
- ▶ ConvertTempPortType_xsdtypes.rpgleinc: Standard data types include
- ▶ ConvertTempPortType.rpgle: RPG web service implementation code
- ▶ ConvertTempPortType.rpgleinc: RPG web service include

From an RPG programmer perspective, the only files that you must look at are the ConvertTempPortType.rpgleinc and ConvertTempPortType_xsdtypes.rpgleinc files. The ConvertTempPortType.rpgleinc file defines the RPG functions to create and destroy a web service interface object, in addition to the web service operations. Also defined in the file are any types that are needed by the web service operations. The ConvertTempPortType_xsdtypes.rpgleinc file defines all the primitive types and various constants.

Look at the contents of the ConvertTempPortType.rpgleinc file, as shown in Figure 5-29. The first thing that you see are data structures for the input and output parameters of the web service operations.

```
D CONVERTTEMPInput_t...
D           DS           qualified based(Template)
D isNil_CONVERTTEMPInput_t...
D           1n
D TEMPIN           likeds(xsd_string)

D CONVERTTEMPResult_t...
D           DS           qualified based(Template)
D isNil_CONVERTTEMPResult_t...
D           1n
D TEMPOUT           likeds(xsd_string)
```

Figure 5-29 Contents of the ConvertTempPortType.rpgleinc file

The xsd_string data type is a primitive data type. All primitive data types are defined in ConvertTempPortType_xsdtypes.rpgleinc. Figure 5-30 shows how xsd_string is defined.

```
D xsd_string...
D           DS           qualified
D           based(Template)
D isNil           1n
D value           a   varying(4) len(128)
D reserved           1a
```

Figure 5-30 xsd_string definition

Continuing on with what is in `ConvertTempPortType.rpglein`, you see functions to create and destroy a handle to the web service stub, as shown in Figure 5-31.

```
D stub_create_ConvertTempPortType...
D          PR          1N  extproc('stub_create_ConvertTempPo+
D          rtType@')
D this          likeds(This_t)

D stub_destroy_ConvertTempPortType...
D          PR          1N  extproc('stub_destroy_ConvertTempP+
D          ortType@')
D this          likeds(This_t)
```

Figure 5-31 Creating and destroying the handle to the web service

The web service is called `ConvertTempPortType`. So, to get an instance of the web service, you call the `stub_create_ConvertTempPortType ()` function. Use the handle that is returned by the function to call the web service operation. To destroy the web service instance, call the `stub_destroy_ConvertTempPortType ()` function.

The next thing that you see are functions that represent the web service operations, as shown in Figure 5-32.

```
D stub_op_converttemp_XML...
D          PR          1N  extproc('converttemp_XML@')
D this          likeds(This_t)
D Value0        likeds(CONVERTTEMPInput_t)
D out           likeds(xsd_string)

D stub_op_converttemp...
D          PR          1N  extproc('converttemp@')
D this          likeds(This_t)
D Value0        likeds(CONVERTTEMPInput_t)
D out           likeds(CONVERTTEMPResult_t)
```

Figure 5-32 Functions that represent the web services operations

The `ConvertTempPortType` port type has two operations that are called `converttemp` and `converttemp_XML`. The corresponding RPG stub operations are `stub_op_converttemp()` and `stub_op_converttemp_XML()`.

Now let us move on to the next step, which is building an application that uses the web service stub.

Creating an application that uses the stub

After the client stub is generated, the stub can be used to create a web service client application. Figure 5-33 on page 115 shows the RPG client application using the created RPG stub to start the `converttemp` web service operation.


```

h DFTNAME (CVTTEMP)
*
/copy ConvertTempPortType.rpgleinc

d OutputText      s           50
d WsStub          ds          likeds(This_t)
d Input           ds          likeds(CONVERTTEMPInput_t)
d Result          ds          likeds(CONVERTTEMPResult_t)

*-----
* Program entry point. The input parameter is a character field
* representing the temperature in Fahrenheit.
*-----
C      *ENTRY      PLIST
C      PARM        TEMPIN      32

/free
// Get a Web service stub. The host and port for the endpoint may need
// to be changed to match host and port of Web service. Or you can pass
// blanks and endpoint in the WSDL file will be used.
clear WsStub;
WsStub.endpoint = 'http://localhost:10000/web/services/ConvertTemp';

clear input;
Input.TEMPIN.value = %trim(TEMPIN);

if (stub_create_ConvertTempPortType(WsStub) = *ON);

// Invoke the ConvertTemp Web service operation.
if (stub_op_ConvertTemp(WsStub:Input:Result) = *ON);
    OutputText = Input.TEMPIN.value + ' Fahrenheit is '
                + Result.TEMPOUT.value + ' Celsius.';
else;
    OutputText = WsStub.excString;
endif;

// Display results.
dsply OutputText;

// Destroy Web service stubs.
stub_destroy_ConvertTempPortType(WsStub);
endif;

*INLR=*ON;
/end-free

```

Figure 5-33 Example RPG program calling the converttemp web service operation

Build the client application by using the commands that are shown in Figure 5-34 from the CL command line.

```

CRTRPGMOD MODULE (CVTTEMP/CNVRTTEMP)
SRCSTMF ('/convtemp/RPG/ConvertTempClientWSDL2RPG.rpgle')

CRTPGM PGM (CVTTEMP/CNVRTTEMP)
MODULE (CVTTEMP/CNVRTTEMP)
BNDSRVPGM (QSYS/DIR/QAXIS10CC CVTTEMP/WSRPG)

```

Figure 5-34 CL commands to build the program by using the web service stubs

The application is built.

The integrated web services client for ILE supports only SOAP-based web services.

More information about integrated web services support can be found at the following website:

<http://www.ibm.com/systems/power/software/i/iws/>

5.4.10 Web services: PHP

This section describes the features of the PHP language that support web services. Because this topic is represented in many other documents, this section focuses on the fundamentals and what makes PHP on IBM i so uniquely posed to deliver incredibly scalable web services. Exploration includes both SOAP and RESTful approaches to web services in addition to the payload typically used in communicating between systems.

Payload: Definition

Payload, as the name implies, is the data that is transferred between systems. As a quick review, web services are a 21st century version of Electronic Data Interchange (EDI). Instead of x.12 as the payload, many web services communicate by using XML or JSON. Either method can be adequate for communicating most web services payload, but there are some fundamental differences. XML can deliver higher levels of accuracy, but at a performance cost because of the verbosity of the payload. The size of the package that is delivered by XML can easily be two to three times the size of JSON and can, therefore, add to the latency of a web service. JSON sacrifices some accuracy in favor of a lighter weight payload. Think of what you might be sending when you pick a method.

PHP impact on web services payload

PHP supports various payload types, including XML and JSON. But PHP can also support the native types of communication such as PHP arrays. The obvious limitation is that you must use PHP on both sides of the web service if you are going to pass a PHP array payload. Given the nature of web services that are trying to make the interconnected world of computer communication cross platform ubiquitous, you might be asking yourself why you might create a web service that generated only PHP arrays. However, many of the program calls of the future will evolve from hardened calls that are in RPG to loosely coupled requests, such as the ones we see in web services inside our own network. Therefore, if you know that there is PHP on both sides of the web service, you can pass a PHP array and enjoy the absolute fastest payload processing time that is available. One of the best implementations, however, allows for the payload to be configurable through the request. For example, a PHP web service client can ask the provisioning web service for a PHP array payload and thus improve the performance of both the server and the client. If, however, the request is denied, the performance impact is a single “if” test.

Other portions of this chapter have described various types of payload. The next section describes PHP and web services.

Simple comparison of XML and JSON

To illustrate the point of simplicity and gloss over detail and verbosity, here is the exact same data that is placed in a JSON format and then in an XML format.

Figure 5-35 on page 117 shows a snippet of JSON.

```
1 {
2   "productNumber":111111,
3   "productName":staples,
4   "productDescription":"Keep papers together",
5   "productBoxQty":5000,
6   "productCaseQty":20,
7   "productMasterQty":10,
8   "productPrice":1.99
9 }
```

Figure 5-35 Example of JSON

Figure 5-36 shows a snippet of XML.

```
1 <SKU>
2   <productNumber>111111</productNumber>
3   <productName>staples</productName>
4   <productDescription>Keep papers together</productDescription>
5   <productBoxQty>5000</productBoxQty>
6   <productCaseQty>20</productCaseQty>
7   <productMasterQty>10</productMasterQty>
8   <productPrice>1.99</productPrice>
9 </SKU>
```

Figure 5-36 Example of XML

As you can see from the preceding simple examples, there is a significant difference in the format of the files. XML is tag-oriented, and JSON relies on more flexible labels and a simple comma delimiter. In this simple example, this is not a major matter. However, look closely at the number of characters in the JSON example (Figure 5-35); there are about 181 with spaces. The XML package (Figure 5-36) has 292 characters, including spaces. The difference of 111 characters represents more than a 50% increase in the JSON payload size. That might not seem like much, but consider what happens when the web service is refreshing the product master from one system to another and there are 5,000 products. Now you have potentially 111 characters times 5,000 products representing an extra 540 KB of data going across the pipe for no reason other than formatting.

Documentation

As always, details about all of the functions that are described in this chapter can be found at <http://www.php.net>, the online documentation for PHP. There is an entire section on SOAP (<http://php.net/manual/en/book.soap.php>), but no section on REST. However, REST is an implementation of RPC and, therefore, the functions that are used to process these requests exist. We repurpose them for this chapter.

Why use PHP: Ease of use

For consumption of a SOAP-based web service, PHP is practically self-documenting. The PHP language is all about growing by the community for the community. So, it is not unusual to find functions in the PHP language that started life as a custom class and grew into the base of PHP. You see similar things with PHP functions around SOAP.

Example application: No limits

The sample application combines many concepts into a pair of PHP scripts. Typically, there is greater separation of roles, but REST and SOAP can exist in the same application. This coexistence again illustrates the incredible flexibility of PHP.

This application is different from the typical IBM i centric application because there is no local data that is used by the application. Web services are used to retrieve data from remote servers and update a database table that can exist as part of an enterprise resource planning (ERP) system. The primary purpose is to retrieve currency rates from a public-facing web service. This level of integration levels the playing field by introducing PHP to an existing application and making IBM i a real-world contender in the cross-platform activities of the 21st century.

REST or SOAP

Web services typically come in two major flavors: REST and SOAP. SOAP is a formal standard and the PHP functions that make it up are defined at <http://www.PHP.net>. REST is an application of a different standard that is called Remote Procedure Call (RPC). Regardless of the approach, web services do a great job of making the communication between two computer systems simple and straightforward. There are advantages to both approaches, but, briefly, RESTful services are typically simple, straightforward requests for simple data elements, and SOAP services usually are complex and can provide for greater precision on data types.

Consider a sample application. First, the sample application display (seen in Figure 5-37) asks the user to select two currency types and an initial value. When selected, the user can click **Get Rate** and the PHP script takes it from there.

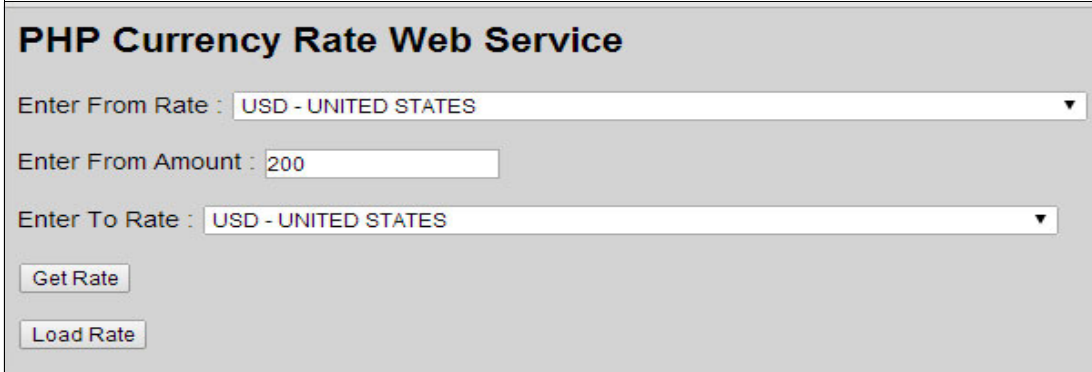


Figure 5-37 User display window for the web service sample application

Example 5-6 shows the PHP code that was used to build the application display and call the various web services based on the action that is taken.

Example 5-6 User display sample PHP form

```
<style>
table {
    border: 4px solid black;    font-size:1em;
    font-family:Arial;background-color: paleturquoise
}
body {
    font-size:1em;
    font-family:Arial;
    background-color: lightgray
}
td, th {
```

```

        border: 1px solid black;
    }
</style>
<?php
    include 'currency1.php';

    if(isset($_POST['calcRate']))
    {
        echo "<h2>Simple Currency Rate Calculator </h2>";
        echo "<hr>";
        $conversionRate = getCurrencyRate($_POST['fromRate'], $_POST['toRate']);
        echo '<table><tr><th>Parm</th><th>Value</th></tr>';
        echo "<tr><td>From Currency</td><td>{$_POST['fromRate']}</td></tr>";
        echo "<tr><td>To Currency</td><td>{$_POST['toRate']}</td></tr>";
        echo "<tr><td>Conversion Rate</td><td>$conversionRate</td></tr></table>";
        $fromAmount = (float) $_POST['fromAmount']; $toAmount = (float) $_POST['toAmount'];
        $conversion = $fromAmount * $conversionRate;
        echo "<br /><br />${fromAmount} {$_POST['fromRate']} is worth $conversion
    {$_POST['toRate']}";
        echo "<hr>";
    }

    if(isset($_POST['loadRates'])) {
        loadRates();
    }

    $option = getCurrencyCode('option');
    sort($option);
    //var_dump($option);

?>

<h2>PHP Currency Rate Web Service</h2>

<form name="currencyForm" method="POST" action="currencyForm.php">
    Enter From Rate :
    <select name="fromRate">
        <?php foreach ($option as $listItem) {
            echo $listItem;
        }?></select><br/><br/>
    Enter From Amount :
    <input name="fromAmount" type="text" value="<?php echo $_POST['fromAmount'];
?>"><br/><br/>

    Enter To Rate :
    <select name="toRate">
        <?php foreach ($option as $listItem) {
            echo $listItem;
        }?></select><br/><br/>
    <input name="calcRate" type="submit" value="Get Rate">
    <br /><br />
    <input name="loadRates" type="submit" value="Load Rate">

</form>

```

REST

RESTful web services are designed to run much like what IBM i developers call an API. RESTful services are direct requests for specific information with little complexity. In the PHP landscape, things are kept simple. The sample application is going to retrieve an XML package that contains a list of currency codes. The XML package is transformed into a PHP array to provide the options in a drop-down list. Green screen applications typically provide for high productivity by keeping the 5250 data stream lightweight. If a user wants to see a list of possible values for an input field, they press F4 and a window opens where they select the value. That is where the performance hit is incurred, as the window pulls the data from DB2 upon request. In the web world, we use drop-down lists, which give the user a similar experience. Drop-down lists are typically populated before the initial page is presented. This means that an application with 20 input fields can run 20+ SQL statements before letting the user see the display.

This application has a function that contains all the code that is necessary to dynamically pull the list of currency codes hot from a web service and return an array of either the raw values or the values that are embedded in an option clause, as shown in Figure 5-38. This configuration gives the caller the flexibility to retrieve the list as they see fit and improve efficiency because they do not need to format the data in the option list, which is arguably the most common request. Later, you look at how to use the data cache to save the retrieved data and thus improve the overall performance of the application.

```
<form name="currencyForm" method="POST" action="currencyForm.php">
  Enter From Rate :
  <select name="fromRate">
    <option value="AED">AED - UNITED ARAB EMIRATES</option>
  <option value="AFN">AFN - AFGHANISTAN</option>
  <option value="ALL">ALL - ALBANIA</option>
  <option value="AMD">AMD - ARMENIA</option>
  <option value="ANG">ANG - CURA&Ccedil;AO</option>
  <option value="ANG">ANG - SINT MAARTEN (DUTCH PART)</option>
  <option value="AOA">AOA - ANGOLA</option>
```

Figure 5-38 REST web services call return data

The **getCurrencyCode()** call, which is shown in Example 5-6 on page 118 and highlighted in bold, makes a RESTful request to a web service by using **file_get_contents()**. The code for this REST web services call is shown in Example 5-7.

Example 5-7 PHP REST base web services code

```
<pre><?php

// Service Functions...
function getCurrencyCode($style = 'array') {
    $currencyCode = 'http://www.currency-iso.org/dam/downloads/table_a1.xml';

    $currencyCodeXML = file_get_contents ( $currencyCode );

    $simpleXML = simplexml_load_string ( $currencyCodeXML );

    foreach ( $simpleXML->CcyTbl->CcyNtry as $codes ) {

        $Ccy = htmlentities ( ( string ) $codes->Ccy );
        if ($Ccy) {
            $CtryNum = htmlentities ( ( string ) $codes->CtryNm );
            if ($style == 'option') {
                $optionString = ' <option value="' . $Ccy . '"';
```

```

        if ($Ccy=='USD' && $CtryNum == "UNITED STATES") //default country...
            $optionString .= 'selected';
        $optionString .= ">$Ccy - $CtryNum</option>\n";
        $option [] = $optionString;
    } else
        $option [] = array (
            'Code' => $Ccy,
            'Country' => $CtryNum
        );
    }
}
if ($option)
    return $option;
else
    return false;
}
function getCurrencyRate($fromRate, $toRate) {
    //$serviceURL = 'http://www.websvcicex.net/currencyconvertor.asmx';
    $serviceWSDL = 'http://www.websvcicex.net/currencyconvertor.asmx?WSDL';

    $currencySoapClient = new SoapClient ( $serviceWSDL );

    $parms = array (
        "FromCurrency" => $fromRate,
        "ToCurrency" => $toRate
    );

    $responseObj = $currencySoapClient->ConversionRate ( $parms );

    $conversionRate = $responseObj->ConversionRateResult;

    return $conversionRate;
}
function loadRates() {
    $conn = db2_connect ( '*LOCAL', 'PHPUSER', 'phpuser1' );
    if (! $conn) {
        echo 'Error connecting to DB2--error code ' . db2_stmt_error () . ' - ' .
        db2_stmt_errormsg ();
        exit ();
    }
    $sql = 'select * from zenddata.rates';
    $stmt = db2_exec ( $conn, $sql );
    if (! $stmt) {
        echo 'Error accessing data--error code ' . db2_stmt_error () . ' - ' .
        db2_stmt_errormsg ();
        exit ();
    }

    print '<table border=2><tr><th>From Currency</th><th>To
Currency</th><th>Rate</th></tr>';
    while ( $row = db2_fetch_assoc ( $stmt ) ) {
        $rate = getCurrencyRate ( $row ['FROMCURRENCY'], $row ['TOCURRENCY'] );
        $today = date ( 'm/d/Y' );
        var_dump ( $today );
    }
}

```

```

    $sql2 = "update zenddata.rates set CONVERSIONRATE = $rate, CONVERTDATE =
' $today'
        where FROMCURRENCY = '{ $row['FROMCURRENCY']}' and
            TOCURRENCY = '{ $row['TOCURRENCY']}'";
    $stmt2 = db2_exec ( $conn, $sql2 );
    echo
" <tr><td>{ $row['FROMCURRENCY']}</td><td>{ $row['TOCURRENCY']}</td><th>$rate</td></t
r>";

    echo 'Table Rates Updated in DB2</table>';
}
}
?></pre>

```

This function can be thought of as a way to capture the content of stream files from the “root” file system of the IFS. A web page is nothing more than a file and the URL is simply the address where the file is. PHP can handle that and, as a result, pulls the content that is provisioned by the web page. This web page in particular happens to be producing not HTML but XML that contains the values of the country codes, which is exactly what we need for the second web service. A sample of the structure of the XML file is shown in Figure 5-39.

```

▼<ISO_4217 Pblshd="2014-01-01">
  ▼<CcyTbl>
    ▼<CcyNtry>
      <CtryNm>AFGHANISTAN</CtryNm>
      <CcyNm>Afghani</CcyNm>
      <Ccy>AFN</Ccy>
      <CcyNbr>971</CcyNbr>
      <CcyMnrUnts>2</CcyMnrUnts>
    </CcyNtry>
    ▼<CcyNtry>
      <CtryNm>ÅLAND ISLANDS</CtryNm>
      <CcyNm>Euro</CcyNm>
      <Ccy>EUR</Ccy>
      <CcyNbr>978</CcyNbr>
      <CcyMnrUnts>2</CcyMnrUnts>
    </CcyNtry>
    ▼<CcyNtry>
      <CtryNm>ALBANIA</CtryNm>
      <CcyNm>Lek</CcyNm>
      <Ccy>ALL</Ccy>
      <CcyNbr>008</CcyNbr>
      <CcyMnrUnts>2</CcyMnrUnts>
    </CcyNtry>
  </CcyTbl>
</ISO_4217 Pblshd="2014-01-01">

```

Figure 5-39 XML that is returned by the `getCurrencyCode` web service

After the web service is requested, we put the resulting XML file into a Simple XML Object. This makes accessing the complex structure of the XML schema easy, as you can see by the `foreach()` loop that reads through it. Each iteration of the `foreach()` produces a Simple XML object from the next member of the original master list, which, again, is part of the elegance of PHP. After the country code and country name are parsed, the values are stored in a PHP array in either raw data format or prepared as an option list.

This function needs to run only once to populate both drop-down lists. The values can also be cached by using the free data cache in Zend Server and thus improve response time even more.

SOAP

The user selects a “from” and a “to” currency from the drop-down lists that are populated by the function that is described in “REST” on page 120. The two values that are selected are passed to another function that calls a SOAP-based web service. The SOAP service takes the inputs and returns the requested conversion rate. For example, a request of the conversion rate from USD (United States Dollars) to GBP (Great British Pound) yields a value of .5973. This value is used to calculate the converted dollar amount for display purposes.

One characteristic of a SOAP-based web service is that it is self-defining because of Web Services Description Language (WSDL). WSDL is typically provisioned and accessed before the web service is called. The level of detail of a WSDL file can be extensive, but this example is simple.

Unlike the RESTful web service, the SOAP service typically has two URLs: one for the WSDL and the other for the service call itself. In this case, there is a built-in class of PHP called SoapClient that can read the WSDL and start communicating with the service. In this example, we coded both URLs, but the SoapClient needs only the WSDL because there is an <address> tag that states the location of the service and that is an ideal place for this value. We coded the URL for documentation purposes and, in a few lines of code, we have a working web service that returns a custom conversion rate between international currencies. In the REST code in Example 5-7 on page 120, the SOAP-based web services URLs are highlighted in bold.

After selecting the from and to currencies, and entering an initial currency amount, the resulting window opens when you click **Get Rate**. Figure 5-40 shows the requested currency types, conversion rate, original amount, and new target currency amount.

The screenshot shows a web interface titled "Simple Currency Rate Calculator". At the top, there is a table with two columns: "Parm" and "Value". The table contains three rows: "From Currency" with value "USD", "To Currency" with value "GBP", and "Conversion Rate" with value "0.5973". Below the table, the text "200 USD is worth 119.46 GBP" is displayed. The main section is titled "PHP Currency Rate Web Service" and contains three input fields: "Enter From Rate" with a dropdown menu showing "USD - UNITED STATES", "Enter From Amount" with a text input field containing "200", and "Enter To Rate" with a dropdown menu showing "USD - UNITED STATES". At the bottom of this section, there are two buttons: "Get Rate" and "Load Rate".

Figure 5-40 PHP Currency Rate web service

Something practical

Having the ability to dynamically request currency codes is useful for the form population. In many accounting systems, there is usually a table in DB2 that is loaded “periodically” to reflect the currency codes rates because they tend to change daily. It can be helpful if the currency code values in the DB2 table can be updated by part of this PHP script process and then scheduled to run in batch. After the code is written to update the DB2 table, the batch submission can be achieved two ways: the first is with a CL program calling the PHP script in Call Level Interface mode, and the second is by using a premium feature of Zend Server called job queue or by using a CL program and php-cli.

The last function in the example application is tied to the Load Rate button and reads through the DB2 table and calls the web service once for each record in the combination of currency codes. The update statement replaces the previous value of the currency code, and you see an example of how to update a DB2 Date field so that the table reflects the date of the last update. Here is the SQL to create the table:

```
CREATE TABLE ZENDDATA.RATES (FROMCURRENCY CHAR (3 ) NOT NULL WITH
DEFAULT, TOCURRENCY CHAR (3 ) NOT NULL WITH DEFAULT, CONVERSIONRATE
FLOAT NOT NULL WITH DEFAULT, CONVERTDATE DATE NOT NULL WITH DEFAULT);
```

```
INSERT INTO ZENDDATA.RATES VALUES('USD', 'GBP', .5913, '01/02/2014')
```

After you call the PHP script, the new values are displayed for the currency types, as shown in Figure 5-41.

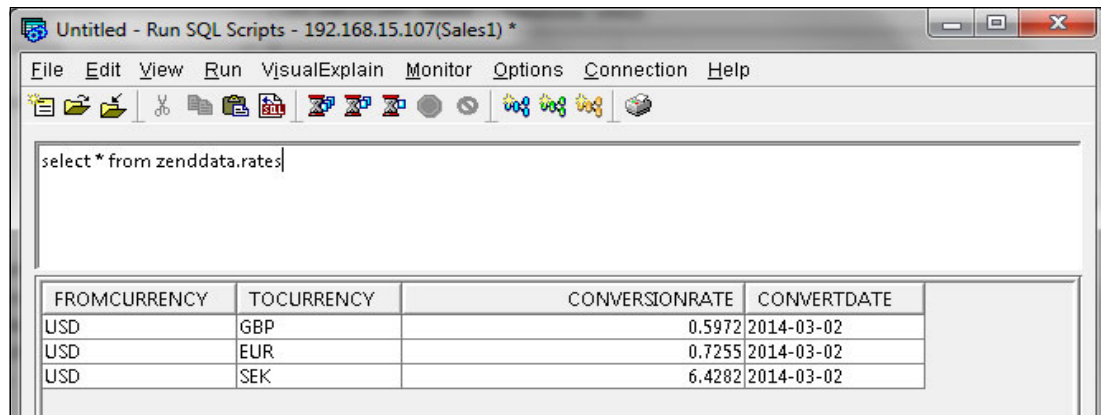


Figure 5-41 Showing the updated currency values after the web services calls

Performance tuning: Data cache

In all versions of Zend Server PHP, there is a feature that is called the data cache. This feature allows the developers to select certain data values and store them in active memory rather than retrieve them from disk. It is useful for frequently accessed elements that do not change often because the data cache can help improve application performance by several factors.

In Example 5-7 on page 120, there is a performance impact each time a web service is called. Many factors beyond your control can influence the performance of the service, such as load and distance. Because these factors cannot be controlled, the data cache can help minimize their impact.

The PHP script function library currency1.php is modified with two minor changes. At the beginning of the getCurrencyCode function, there is a test to see whether the data cache exists. If it does exist, the value is loaded into \$option and then returned by the function and thus the web service is not called. If the value does not exist, the code to populate the cache comes in just after the web service is called and the \$option array is built and loaded for 10 hours (10 * 3600 seconds per hour). These changes are shown in the updated script in Example 5-8.

Example 5-8 Updated web services example using the data cache option

```
<pre><?php

// Service Functions...
function getCurrencyCode($style = 'array') {
```

```

// check to see if the cache exists...
$option = zend_shm_cache_fetch ( 'currencycodes' );
if ($option === null) {

    $currencyCode = 'http://www.currency-iso.org/dam/downloads/table_a1.xml';

    $currencyCodeXML = file_get_contents ( $currencyCode );

    $simpleXML = simplexml_load_string ( $currencyCodeXML );

    foreach ( $simpleXML->CcyTbl->CcyNtry as $codes ) {

        $Ccy = htmlentities ( ( string ) $codes->Ccy );
        if ($Ccy) {
            $CtryNum = htmlentities ( ( string ) $codes->CtryNm );
            if ($style == 'option') {
                $optionString = ' <option value="' . $Ccy . '"';
                if ($Ccy == 'USD' && $CtryNum == "UNITED STATES") // default
country...
                    $optionString .= 'selected';
                $optionString .= ">$Ccy - $CtryNum</option>\n";
                $option [] = $optionString;
            } else
                $option [] = array (
                    'Code' => $Ccy,
                    'Country' => $CtryNum
                );
        }
    }

    // Store the results in cache
    zend_shm_cache_store ( 'currencycodes', $option, 10 * 3600 );
}
if ($option)
    return $option;
else
    return false;
}
function getCurrencyRate($fromRate, $toRate) {
    // $serviceURL = 'http://www.websvcx.net/currencyconvertor.asmx';
    $serviceWSDL = 'http://www.websvcx.net/currencyconvertor.asmx?WSDL';

    $currencySoapClient = new SoapClient ( $serviceWSDL );

    $parms = array (
        "FromCurrency" => $fromRate,
        "ToCurrency" => $toRate
    );

    $responseObj = $currencySoapClient->ConversionRate ( $parms );

    $conversionRate = $responseObj->ConversionRateResult;

    return $conversionRate;
}

```

```

}
function loadRates() {
    $conn = db2_connect ( '*LOCAL', 'PHPUSER', 'phpuser1' );
    if (! $conn) {
        echo 'Error connecting to DB2--error code ' . db2_stmt_error () . ' - ' .
db2_stmt_errormsg ();
        exit ();
    }
    $sql = 'select * from zenddata.rates';
    $stmt = db2_exec ( $conn, $sql );
    if (! $stmt) {
        echo 'Error accessing data--error code ' . db2_stmt_error () . ' - ' .
db2_stmt_errormsg ();
        exit ();
    }

    print '<table border=2><tr><th>From Currency</th><th>To
Currency</th><th>Rate</th></tr>';
    while ( $row = db2_fetch_assoc ( $stmt ) ) {
        $rate = getCurrencyRate ( $row ['FROMCURRENCY'], $row ['TOCURRENCY'] );
        $today = date ( 'm/d/Y' );
        var_dump ( $today );
        $sql2 = "update zenddata.rates set CONVERSIONRATE = $rate, CONVERTDATE =
'$today'
                where FROMCURRENCY = '{$row['FROMCURRENCY']}' and
                TOCURRENCY = '{$row['TOCURRENCY']}'";
        $stmt2 = db2_exec ( $conn, $sql2 );
        echo
"<tr><td>{$row['FROMCURRENCY']}</td><td>{$row['TOCURRENCY']}</td><th>$rate</td></t
r>";

        echo 'Table Rates Updated in DB2</table>';
    }
}
?></pre>

```

There are two performance benefits to this approach. The first benefit is that you can now reduce the number of times the web service is called. If the typical time to call the service is two seconds and you reduce that number by a significant amount because of heavy usage of the GUI, you have saved not only time but also reduced the load on DB2, leaving more DB2 resources for other processes on the system. The second benefit is that you loaded a field in the PHP form directly from memory to significantly improve response and thus improve the productivity of the user.

There are a few additional benefits to this approach:

- ▶ This element is not just available for the user who loads the cache but for every user on the system. It is a little like a file in QTEMP that everyone on the system can share.
- ▶ Each element on the window can be cached and each can have its own expiration interval. For example, say that you had an order entry window with 20 elements and 10 of those elements have drop-down lists that do not change often. Even at 100 milliseconds per list, you are saving an entire second of response time, and that can make the difference between a positive and not so positive user experience.

- The elements can be forced to expire programmatically or they can be cleared by an administrator by clicking **Clear Cache** in the upper right corner of the Data Cache component in the Zend Server administration GUI (shown in Figure 5-42).

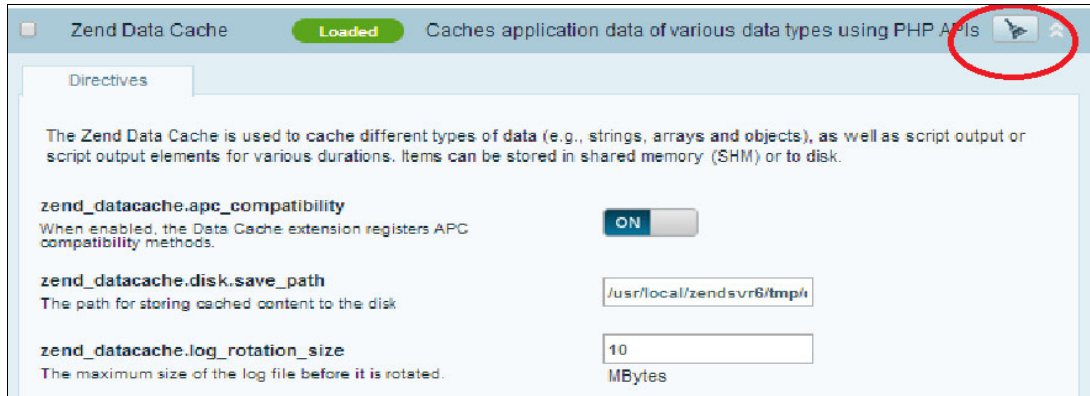


Figure 5-42 Cache component of the Zend Server admin GUI

WSDL cache

In the `getCurrencyRate` function, there are two requests of the SOAPClient each time you run the request. One request comes when the WSDL is retrieved and another when the service is called. But how often does the WSDL change? As with EDI, many web services have SLAs and rules for operation. Currency rates might change wildly throughout the day, but the WSDL itself might not change for months or years. For this reason, the PHP community built another feature into the server that reduces the number of times that the WSDL is retrieved. This system-level setting can cache the WSDL for a significant amount of time and can have a dramatic impact on PHP performance because many unnecessary requests are thrown away in favor of highly improved response times. The cache can be cleared programmatically by using the `ini_set` function to turn it off, if needed. For example:

```
ini_set("soap.wsdl_cache_enabled", "0");
```

This setting is controlled through the Zend Server administration GUI interface, as shown in Figure 5-43.

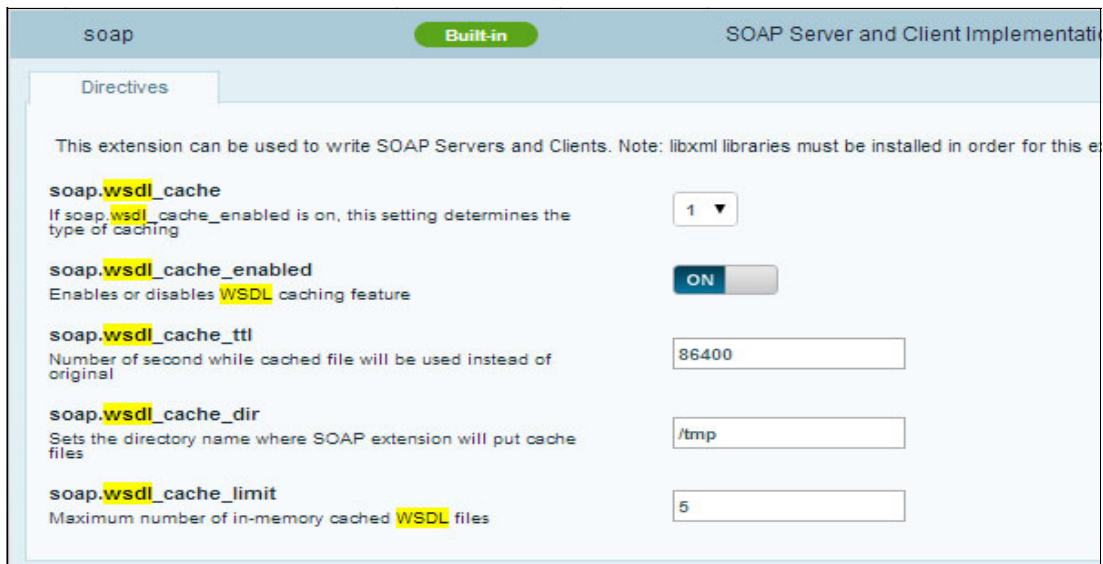


Figure 5-43 WSDL cache settings within the Zend Server administration GUI

PHP provides many benefits in some small packages. These scripts are calling multiple services with a database update and barely 100 lines of code were written for a full application example. This provides a concise basis for developing PHP applications by using the procedural approach to PHP. This application might have been written using an object-oriented approach and possibly should have been. However, as many IBM i developers have a solid background in procedural such as RPG and COBOL, it is good to know that the developer can grow into the object-oriented model gradually. It is called modernization on your terms.

5.4.11 REST Web Services: XML Service

XML Service support is built on XML. It is a good fit for REST-based web services. A more detailed explanation is provided in “XMLSERVICE HTML/XML interface” on page 140. The XML Service support can receive REST-based web services requests that can then process your back-end ILE programs or access your DB2 for i data. For more details, you can also see the XML Service website:

<http://youngiprofessionals.com/wiki/index.php/XMLSERVICE/RPGWebServices>

5.5 Cross environment

This section describes some of the methods and options that are available to you when you are developing in an environment that has multiple systems. Typically, this process involves the interface in one place needing to access data or a service in another place.

5.5.1 Cross environment: Java to ILE

Other sections of this publication describe ways to expose ILE programs through industry-standard mechanisms, such as XML or JSON-based web services, SSH invocations, and TCP/IP communications. In general, Java applications can communicate with these industry-standard protocols easily. Beyond that, however, there are many other ways that Java can call or interact with ILE code. This section describes some of the ways that Java to ILE integration is accomplished.

IBM Toolbox for Java

The IBM Toolbox for Java is a popular way to call ILE from Java. The toolbox provides APIs that quickly enable Java programs to access much information on IBM i. This section describes techniques for integrating ILE functions into a Java application, but the toolbox can do much more than that. All of these tasks (and then some) can be easily done with the toolbox:

- ▶ Run commands.
- ▶ Call programs.
- ▶ Access DB2 for i through JDBC or record-level access.
- ▶ Access IFS files, spooled files, data queues, environment variables, system values, users, jobs, user spaces, and much more.

As you can imagine, the toolbox is useful for various tasks, from basic functions to system management. Also, because the toolbox enables all of these functions for TCP/IP connected Java applications, it allows flexibility for remote or multitier implementations. Many IBM products are built on top of the IBM Toolbox for Java.

The IBM Toolbox for Java is part of the JTOpen family, which can be downloaded at <http://jt400.sourceforge.net>. It is also available (at no additional charge) with the JC1 licensed program on IBM i, which installs the necessary JAR files into the /QIBM/ProdData/HTTP/Public/jt400/lib/ directory. The files of interest are jt400.jar, the standard cross-platform APIs, and jt400Native.jar, which contains the same APIs with optimizations for running on IBM i.

A brief introduction to using the toolbox

Connections are acquired through the AS400 class (Example 5-9). From a code perspective, the AS400 class serves as the starting point for nearly everything (because nearly everything requires a connection). Example 5-9 shows several ways to acquire an AS400 object. Required properties (host name, user ID, and password) can be specified on the constructor or explicitly set after the object is created.

Example 5-9 AS400 class examples

```
AS400 as400 = new AS400();
as400.setSystemName("mySystem");
as400.setUserId("myUID");
as400.setPassword("myPWD");
AS400 as400_2 = new AS400("mySystem", "myUID", "myPWD");
```

When you have an AS400 object, it can typically be used to construct other types of objects. For example, here is how you can perform a simple command call. In this example, we use the AS400 object to create a CommandCall (Example 5-10) object, which has the necessary methods.

Example 5-10 CommandCall example

```
CommandCall cc = new CommandCall(as400);
try {
    boolean isSuccessful = cc.run("CRTLIB FRED");
    // check isSuccessful to see if the command ran successfully
    for(AS400Message msg : cc.getMessageList()){
        // do something with the messages
    }
} catch (Exception e) {
    log(e);
}
```

Using the toolbox's ServiceProgramCall class

One of the classes that are shipped with the toolbox is aptly named ServiceProgramCall. Not surprisingly, it enables a Java program to call entry points in ILE service programs. To do so, you must know the types and purpose of the input and output parameters. Example 5-11 calls an entry point that has a zoned decimal as an input parameter, and one as an output parameter, both of which are passed by reference.

Example 5-11 ServiceProgramCall class example

```
// Create a list to hold input parameters.
ProgramParameter[] parmList = new ProgramParameter[2];

// Input parameter
AS400ZonedDecimal tempVar = new AS400ZonedDecimal(10, 2);
// Output parameter
AS400ZonedDecimal outVar = new AS400ZonedDecimal(10, 2);
```

```

// Save parameters in the parameters list
// "inParam" is a BigDecimal object passed in to this function as a parameter.
parmList[0] = new
ProgramParameter(ProgramParameter.PASS_BY_REFERENCE,tempVar.toBytes(inParam));
parmList[1] = new ProgramParameter(ProgramParameter.PASS_BY_REFERENCE,10);

// Create a service program
ServiceProgramCall svcpgm = new ServiceProgramCall(conn);
// Set the fully qualified service program and the parameter list.
svcpgm.setProgram("/QSYS.lib/FLGHT400C.lib/SMPLAPISVC.srvpgm",parmList);

// Set the procedure to call in the service program.
svcpgm.setProcedureName("CONVERTTEMP");

// Invoke the program
if (svcpgm.run() == false) {
    // If program invocation failed, process messages
    AS400Message[] msgs = svcpgm.getMessageList();
    for (int i = 0; i < msgs.length; ++i) {
        // for demo/debugging purposes
        // In a production-level application, return
        System.out.println(msgs[i].getID() + ": " +
            msgs[i].getText());
    }
} else {
    // Get the value of the output parameter
    byte[] outVar1 = parmList[1].getOutputData();
    // the toObject() method of the AS400ZonedDecimal class re-returns a BigDecimal
    object
    BigDecimal outParam =
    (BigDecimal)outVar1.toObject(parmList[1].getOutputData());
    // do something with the output here
}

```

In Example 5-11 on page 129, the input and outputs to the service program are zoned decimals. To expose this aspect to Java developers, the toolbox includes an AS400ZonedDecimal class. There are many other classes that are specific to other data types, including AS400Array, AS400Bin4, AS400ByteArray, and AS400Text. While the AS400ZonedDecimal class maps a zoned decimal to the Java BigDecimal, other classes map the data to appropriate data types, as shown in Figure 5-44 on page 131. The toolbox handles code page, byte order, and data conversion issues.

Java data type		IBM i data type
Object[]		Array
short	↔	2 byte binary
int		2 byte unsigned binary
int	↔	4 byte binary
long		4 byte unsigned binary
long	↔	8 byte binary
byte[]		Byte array
float	↔	4 byte floating point
double		8 byte floating point
BigDecimal	↔	Packed decimal
BigDecimal		Zoned decimal
Object[]		Structure
String	↔	Text

Figure 5-44 Java to IBM i data type conversions

ServiceProgramCall with PCML

In Example 5-11 on page 129, we incorporated all of the data type knowledge into the Java code.

Alternatively, when you build the ILE parts, you can specify *PCML output, which generates an XML-like representation of the input and output parameters for the service program. In Example 5-12, the PCML represents the service program that we are calling.

Example 5-12 *PCML example

```
<pcml version="4.0">
<program name="CONVERTTEMP" entrypoint="CONVERTTEMP"
    path="/QSYS.lib/FLGHT400C.lib/SMPLAPISVC.srvpgm">
    <data name="TEMPIN" type="zoned" length="10"
precision="2" usage="input" />
    <data name="TEMPOUT" type="zoned" length="10"
precision="2" usage="output" />
    </program>
</pcml>
```

The toolbox can use this PCML data to simplify calls to this service program. In Example 5-13, the ProgramCallDocument class is used to load the PCML and call the entry point of interest. Unlike in the previous Java example (Example 5-12), the specific knowledge of the input and output parameters is encapsulated in the PCML instead of into the Java code itself. This makes the Java code smaller and can make maintenance easier. If the service program input and output types change, the Java program can use the updated PCML, instead of requiring the Java code to be changed to the new specification.

Example 5-13 ProgramCallDocument example

```
// Create a Program Call document
ProgramCallDocument pcmlDoc =
new ProgramCallDocument(conn,
"com.ibm.sample.pcall.simpleapi");
```

```

// Set input parameters and invoke the program
// "inParam" is a BigDecimal object passed in to this function as a parameter.
pcmlDoc.setValue("CONVERTTEMP.TEMPIN", inParam);
boolean result = pcmlDoc.callProgram("CONVERTTEMP");

if(result == false){
    // Program did not run successfully
    AS400Message[] msgs = pcmlDoc.getMessageList("CONVERTTEMP");
    for (int i = 0; i < msgs.length; ++i) {
        // for demo/debugging purposes
        // In a production-level application, return
        System.out.println(msgs[i].getID() + ": " + msgs[i].getText());
    }
}else{
    // Program ran successfully
    BigDecimal outParam =
    (BigDecimal) pcmlDoc.getValue("CONVERTTEMP.TEMPOUT");
    // do something with the output here
}

```

ProgramCall

To call ILE programs directly, you can use a ProgramCall class. As you might guess, it is like ServiceProgramCall, except that it calls a program instead of a service program. The calling conventions and data type classes (like AS400ZonedDecimal) are the same. ProgramCall can also use PCML.

Calling SQL stored procedures through JDBC

One of the ways to get at ILE code is by referencing it in a database context. You can access the database with JDBC, which is described in 5.5.4, "Database access" on page 163. Although JDBC enables you to use SQL **SELECT** and **INSERT** statements, it also allows you to do much more, including the invocation of SQL stored procedures. The usage of stored procedures has several advantages over the ProgramCall and ServiceProgramCall interfaces that are provided with the toolbox. For one, the Java application can pass in up to 1,024 parameters, compared to 35 and seven for ProgramCall and ServiceProgramCall. Also, a stored procedure can return primitive parameters or a variable size result set. Using the ILE function in this manner requires knowledge of SQL and stored procedures.

Using Data Queues

Data Queues serve as another common technique for interfacing Java with ILE programs. They can facilitate inter-program communication between programs of any language. Because the IBM Toolbox for Java allows Java programs to read and write to data queues, it enables Java to take advantage of this powerful instrument.

The data queue function of the toolbox is robust. It supports both sequential and keyed data queues. With a sequential data queue, you can configure the order of reading messages either as LIFO or as FIFO. Also, the APIs provide methods to look at the data queue without removing the item. If the asynchronous nature of data queues is appropriate for your needs, the data queue support might be a good fit.

Example 5-14 shows how to write data to an IBM i data queue by using the IBM Toolbox for Java.

Example 5-14 Calling a Data Queue example

```
AS400 conn = // get AS400 object here
// Create a record format with two fields:
// customer number (4-byte number and name (50-char String)
BinaryFieldDescription custNumberField = new BinaryFieldDescription(new
AS400Bin4(), "CUSTOMER_NUMBER");
CharacterFieldDescription custNameField = new CharacterFieldDescription(new
AS400Text(50, conn), "CUSTOMER_NAME");

RecordFormat sampleRecord = new RecordFormat();
sampleRecord.addFieldDescription(custNumberField);
sampleRecord.addFieldDescription(custNameField);

// Create the data queue object
DataQueue dq = new DataQueue(conn, "/QSYS.LIB/FLGHT400C.LIB/CUSTINFO.DTAQ");

// This will create a Data Queue when the program is run for first time
try{
    // Create a data queue with a maximum size of an entry equal to 96
    dq.create(96);
// Ignore Exception - the Data Queue already exists
} catch (ObjectAlreadyExistsException e) {};

// Create a record based on the record format and populate it
Record data = new Record(sampleRecord);
data.setField("CUSTOMER_NUMBER", new Integer(customerNumber));
data.setField("CUSTOMER_NAME", customerName);

// Convert the record into a byte array that will be written to the Data Queue
byte [] byteData = data.getContents();

// Write the record to the data queue.
dq.write(byteData);
```

JTOpenLite

The IBM Toolbox for Java is by far the most popular member of the JTOpen family. It has been around for several years and is the *de facto* standard for accessing IBM i from Java. However, JTOpen also ships a lightweight counterpart that is known as JTOpenLite. This package, which is in the `jtopenlite.jar` file, provides a different, more lightweight set of APIs for accessing IBM i. Although the API set is not as robust as the IBM Toolbox for Java, it still lets you access programs, service programs, IFS, and commands. It also has a lightweight JDBC driver and record level access support.

JTOpenLite might be the correct choice for a few reasons. The JAR file is smaller, and it tends to have a smaller memory footprint. Several operations also perform faster with this lean offering. Most notably, however, JTOpenLite can be used for creating native Android applications.

Example 5-10 on page 129 demonstrated how to run a command by using the IBM Toolbox for Java. Example 5-15 shows how to perform the same task with JTOpenLite.

Example 5-15 Call Command example with JTOpenLite

```
CommandConnection cc = CommandConnec-tion.getConnection("mySystem", "myUID",
"myPWD");
CommandResult result = cc.execute("CRTLIB FRED");

boolean isSuccessful = result.succeeded();
// check isSuccessful to see if the command ran successfully
outputPrintStream.println("Success? "+);
for(Message msg : result.getMessages()){
    // do something with the messages
}
```

Java Native Interface

The most powerful way to merge Java and ILE is to use the Java Native Interface (JNI). The JNI allows you to write Java native methods in ILE, which can be called from your Java application just like any other Java method. Further, the ILE native code can interact with the currently running Java virtual machine (JVM). From ILE, you can call Java methods, construct new Java objects, set array elements, and more. This integration technique is more complex than the other solutions that are described in this chapter. For more information about ILE native methods for Java, go to the following website:

<http://pic.dhe.ibm.com/infocenter/iseriess/v7r1m0/index.jsp?topic=%2Frzaha%2Frzahai1enativemethods.htm>

Runtime.exec()

Lastly, Java itself offers a fast way of starting jobs: **Runtime.exec()**. This standard (but inherently platform-dependent) mechanism in the Java runtime library starts a command in a new process, as shown in Example 5-16. You can then access the standard input, output, and error streams of this new (child) process. Because Java on IBM i runs in PASE, the **Runtime.exec()** function runs PASE commands. Therefore, you can use the PASE “system” utility to call CL commands. If advanced parameter handling is not needed, and the application always runs on IBM i itself, this is perhaps the quickest and easiest way to run an ILE program from Java.

Example 5-16 Runtime.exec() example

```
Process p = Runtime.getRuntime().exec("system \"CALL PGM(MYPROGRAM/MYLIB)\");
// the standard input of the child process.
OutputStream stdin = p.getOutputStream();
// the standard output of the child process
InputStream stdout = p.getInputStream();
// the standard error of the child process
InputStream stderr = p.getErrorStream();
```

5.5.2 Cross environment: XMLSERVICE

XMLSERVICE is an open source RPG project that was created for web programming simplicity and deployment flexibility for any language and any transport to avoid complexities of dealing with large package web services. Internally, XMLSERVICE is designed with Plain Old XML (POX), which enables a simple REST XML protocol, which avoids the complexity of SOAP/WSDL-based web services. The XMLSERVICE engine is 100% RPG. It never requires Java, PHP, or any other front-end language to serve your IBM i RPG programs to the web. In fact, XMLSERVICE is so simple, that you can almost run your entire IBM i system using only simple HTML/XML.

XMLSERVICE, as its name implies, enables XML services on your IBM i system. The XMLSERVICE library is a collection of open source RPG modules that allow you to access anything on your IBM i machine, assuming you have the correct user profile authority. XMLSERVICE accepts XML documents that contain actions/parameters (<pgm>, <cmd>, <sh>, <sql>, and so on) that perform the requested operations on the IBM i, and then send an XML document of results back to the client. This simple concept, which is shown in Figure 5-45, has become a powerful tool for scripting language clients on IBM i, including Zend Server PHP Toolkit and PowerRuby Toolkit. However, PHP and Ruby are not unique to XMLSERVICE. XML is a universal protocol that is supported by nearly every language, so nearly any language can use XMLSERVICE and almost any transport between the client and the server.

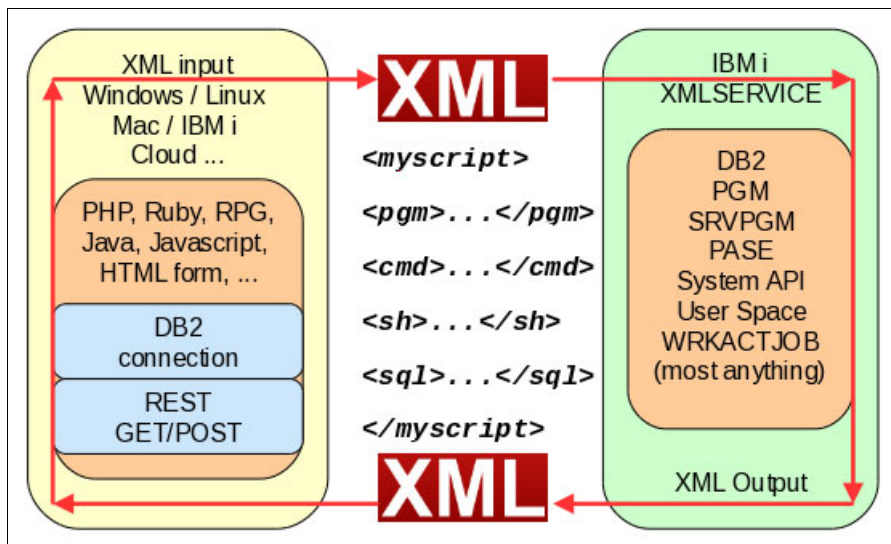


Figure 5-45 XMLSERVICE

Operation and transports

XMLSERVICE XML documents can be transported between IBM i and clients over any connection, in any language. Here are some items of note regarding XMLSERVICE:

- ▶ XMLSERVICE is a single library of open source RPG code that enables XML scripting calls of IBM i resources.
- ▶ The XMLSERVICE RPG engine source can be download from the Yips XMLSERVICE website:

<http://youngprofessionals.com/wiki/index.php/XMLSERVICE/XMLSERVICE>

- ▶ The XMLSERVICE introduction and working examples can be found at Yips XMLSERVICE Introduction website:
<http://174.79.32.155/wiki/index.php/XMLSERVICE/>
- ▶ The XMLSERVICE RPG server library does not require other language products to run on IBM i. However, language teams often provide a client toolkit to greatly simplify XML calls to XMLSERVICE.

Installation

XMLSERVICE is much like the Java toolbox. It is an open source project that is supported by IBM. There are two ways to access the XML Service engine:

- ▶ In true open source fashion, you can download the source from the open source website and compile it.
- ▶ You can use the pre-compiled XML Service engine, which is delivered through the HTTP PTF Group.

The benefit of the open source method is that you are ensured of getting the latest support. The version that is delivered in the PTF group is refreshed from time to time, but still might be out of date. (This approach is the same approach as is taken with the Java toolbox: If you want the latest version, you need to get from the open source site.)

Open source installation

The open source installation is easy for a practiced IBM i developer. First, you must download XMLServices-RPG.zip from the following website:

<http://youngiprofessionals.com/wiki/index.php/XMLSERVICE/XMLSERVICE>

Then, complete the following steps:

1. Extract xmlservice-rpg.zip to xmlservice.savf.
2. Run **crtSAVF FILE(QGPL/XMLSERVICE)**.
3. Use FTP to move xmlservice.savf to QGPL/XMLSERVICE (binary FTP).
4. Run **RSTLIB SAVLIB(XMLSERVICE) DEV(*SAVF) SAVF(QGPL/XMLSERVICE) MBROPT(*ALL) ALWOBJDIF(*ALL)**.
5. Run **ADDLIBLE XMLSERVICE**.
6. Run **XMLSERVICE: test library XMLSERVICE**. This command is good for trying new versions.
7. Run **CRTCLPGM PGM(XMLSERVICE/CRTXML) SRCFILE(XMLSERVICE/QCLSRC)**.
8. Run **call crtxml -- XMLSERVICE library only**.
9. Run **CHGAUT OBJ('/qsys.lib/XMLSERVICE.lib') USER(QTMHHTTP) DTAAUT(*RWX) OBJAUT(*ALL) SUBTREE(*ALL)**.

Note: Other CLPs exist for other team production libraries. XMLSERVICE PGM objects are coded to compile a library only. Therefore, you cannot SAV/RST PGM, and so on, to other libraries.

IBM PTF method

IBM i traditionalists who are comfortable with “supported software” where no compile is required can choose IBM i PTFs to avoid downloading and compiling XMLSERVICE. However, XMLSERVICE source code, tests (ZZCALL), and non-production extras are not included with IBM i PTFs because of size restrictions. Therefore, you can download the following compressed files for tests and other non-production extras that are referenced in this section.

- ▶ V7R1 PTF SI50835 (1.8.1) replaces V7R1 PTF SI48830 (1.7.5).
- ▶ V6R1 PTF SI50834 (1.8.1) replaces V6R1 PTF SI48831 (1.7.5).
- ▶ V5R4 PTF SI48440 (1.7.5) has no additional PTFs planned.

Note: The IBM PTF library is QXMLSERV (not the XMLSERVICE library).

Operation and transports

XMLSERVICE XML documents can be transported between IBM i and clients over any connection and any language. The XMLSERVICE library includes language transports for REST and DB2 connections, which fulfill needs for most internet service applications. These APIs are what allow you to connect from almost anywhere. Figure 5-46 shows the different transports that are available and the different places where a request can originate. The request is sent over the internet. The IBM i jobs (in the middle of the diagram) catch that request and process the XML. Based on the request, the XMLSERVICE job can employ a stateless or stateful connection for communication with that endpoint client.

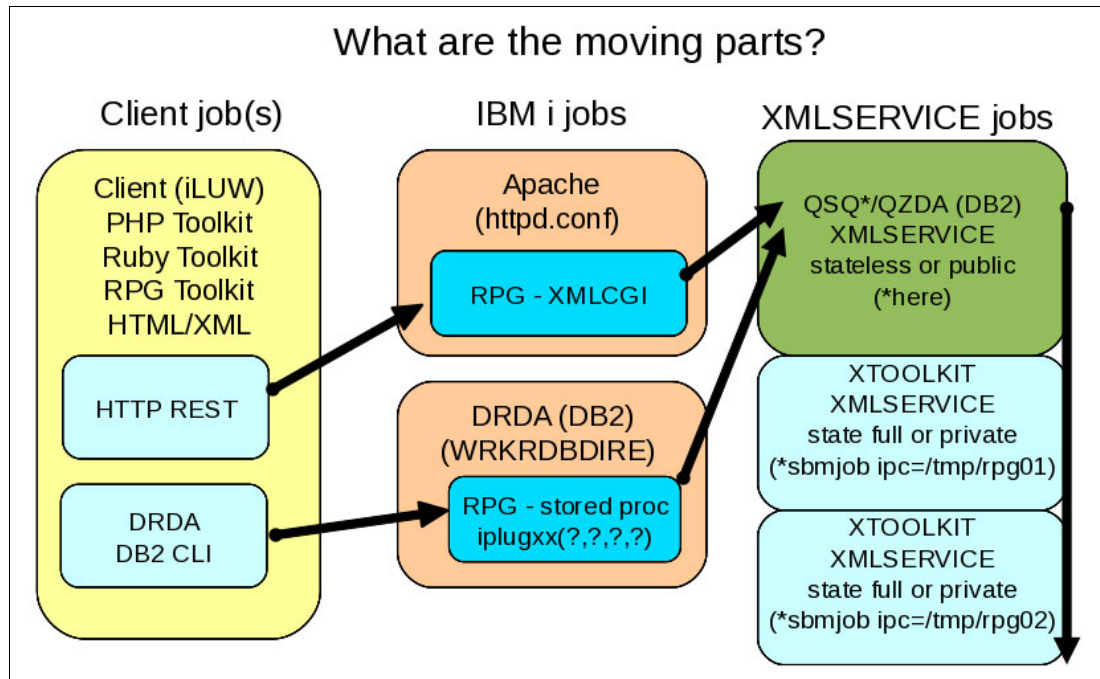


Figure 5-46 XMLSERVICE transports

The XMLSERVICE contains a comprehensive list of APIs:

▶ XMLSERVICE/XMLCGI.PGM

This is the RPG CGI HTTP/REST method GET or POST (traditional web service interface), as shown in Example 5-17.

Example 5-17 RPG CGI HTTP/REST method GET or POST

```
http://myibmi/cgi-bin/xmlcgi.pgm?db2=x@uid=x@pwd=x@ipc=x@ctl=x@xmlin=x@xmlout=x
```

▶ XMLSERVICE/XMLSTOREDP.SRVPGM

- This is an RPG DB2 stored procedure (the IBM premiere DB2 for i).
- There are DB2 drivers local/remote with stored procedure IN/OUT capabilities (traditional DB2 interface).

▶ DB2 drivers local/remote without stored procedure IN/OUT capabilities (loop fetch required)

▶ XMLSERVICE/XMLSTOREDP.SRVPGM

- This is optional custom transport (programmers only).
- If included, and XMLSERVICE transports do not fill your need, you can create your own (sockets, data queues, FTP, and so on). Multiple entry APIs exist in XMLSERVICE that you might find useful.

IBM i CCSID 65535

XMLSERVICE REST and DB2 connections have implicit CCSID conversion between client and server, so your XMLIN/XMLOUT XML document is implicitly CCSID converted by the transport layer. The XMLSERVICE should work. However, IBM i CCSID 65535 (hex) destroys the entire XMLSERVICE scheme. You might see hangs, junk data, dead processes, and so on. Take action on your IBM i to use with the modern ASCII world (all clients, all new scripting languages, and remote/local, including PASE).

Here are some corrective IBM i suggestions (CCSID 37 is an example). Remember, you must end current running jobs and restart for the CCSID changes to enable (including XTOOLKIT jobs):

▶ Change the system CCSID:

```
CHGSYSVAL SYSVAL(QCCSID) VALUE(37)
```

▶ Change the Apache instances (/www/instance/httpd.conf):

```
# protect FastCGI against bad CCSID (dspsysval qccsid 65535)
DefaultFsCCSID 37
CGIJobCCSID 37
```

▶ Change user profiles:

```
CHGUSRPRF USRPRF(FRED) CCSID(37)
```

Connection public or private

XMLSERVICE parameters CTL and IPC enable two types of connections:

- ▶ Public connection. This connection is XMLSERVICE stateless, can run many jobs, and every usage is a fresh start.
 - It is “public” because any IBM i profile usage is a fresh start.
 - Its life scope is the life of a single XML IN/OUT request.

- ▶ Private connection. This connection is XMLSERVICE stateful, can run a single job, and a given profile is routed to XTOOLKIT jobs.
 - CTL='*sbmjob'. If not running, submit a new XTOOLKIT job.
 - IPC="/tmp/myjob1". All requests of IPC route to the XTOOLKIT job (*sbmjob).
 - It is “Private” because one IBM i profile owns the XTOOLKIT job. Any other IBM i profile fails to attach, and must use a new job.
 - Life scope. The scope is forever until ended by an operator or a user program ends the XTOOLKIT job (like the 5250 behavior).

Web programming style (public and stateless)

XMLSERVICE public connections are used when an IBM i resource can be called once by many different user profiles, current data is returned, and no lasting persistent data is needed by the IBM i resource. The programmer meta description of transient resource services is “stateless programming”. In a public or stateless environment, each request that comes into the IBM i from the clients creates a connection. The biggest issue for a public connection is that it is not a high-performance usage of XMLSERVICE. Because every request creates a connection, it is clean and robust, but can cause a bottleneck if too many repetitive requests for the same user come in. Figure 5-47 shows the flow for a public request.

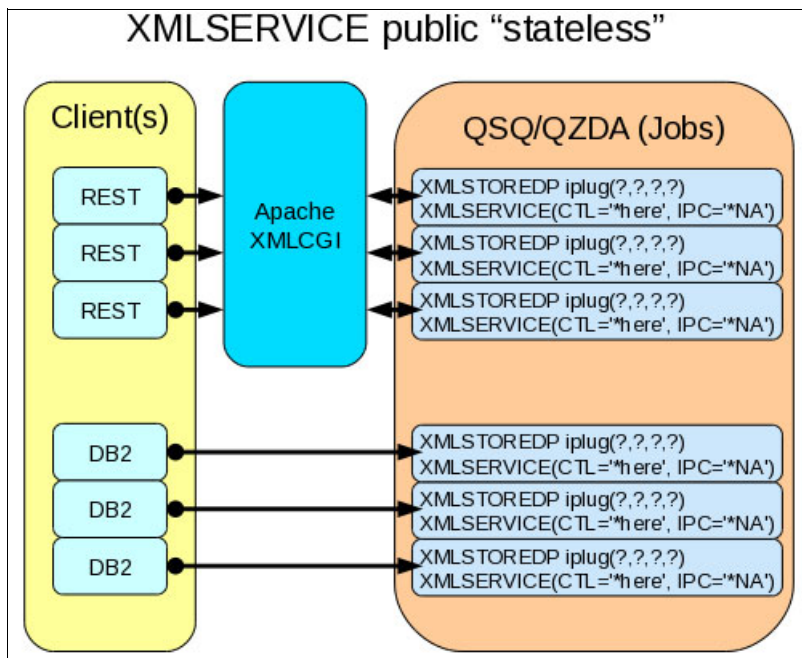


Figure 5-47 XMLSERVICE public request

Traditional programming style (private and stateful)

XMLSERVICE private connections are used when an IBM i resource is called many times by the same user profile, and lasting persistent data is needed by the IBM i resource (RPG variables, open files, and so on). The programmer meta description of required data services is “stateful programming”. As shown in Figure 5-48, each connection that comes in for a certain user is always routed to the same XTOOLKIT job. This can be thought of in a similar manner as a prestart job. The resource is waiting for work.

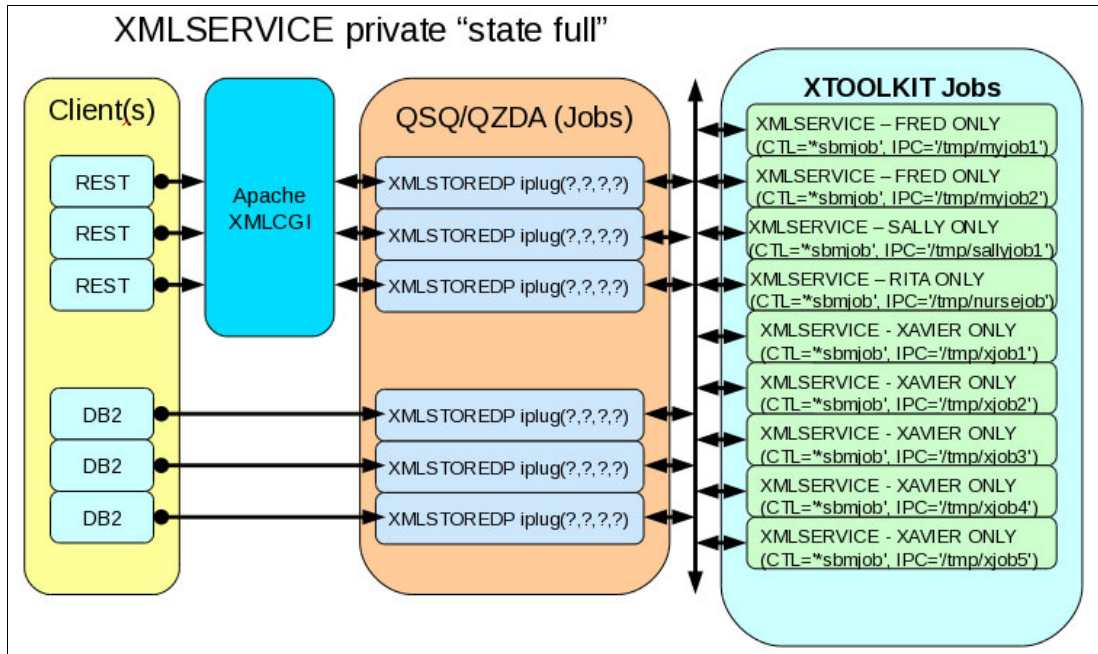


Figure 5-48 XMLSERVICE private connection

Traditional RPG programs usually must track local variables, open files, and so on, between requests, both for correct functioning and performance. The XTOOLKIT jobs that result from CTL='*sbmjob' and IPC="/tmp/xxxx" are similar to 5250 jobs, where a user profile signs on, then uses 5250 job to run other programs (also known as XMLSERVICE design). Also, like multiple 5250 sessions (PC emulators), many different XTOOLKIT jobs can be used by the same user profile.

XMLSERVICE HTML/XML interface

Using your IBM i and XMLSERVICE, without writing one line of code in any language, you can check XMLSERVICE functions by using HTML/XML forms. By any standard, the following XMLSERVICE example is clearly REST web services with no SOAP, no WSDL, no Java, no PHP, no Ruby, or anything but some HTML/XML. In this example, we create a simple web page that is going to submit an XMLSERVICE request to your IBM i system. It is going to perform a SQL request and acquire data from a DB2 database and return the content.

Preparing IBM i

If you want IBM i to listen for a REST request, you can update your Apache server to enable the XML SERVICE REST support. This task is accomplished by adding the following information to your Apache server. For this example, we added the necessary directives to the APACHEDFT server (Example 5-18).

Example 5-18 XML SERVICE REST added to the Apache server

```
/www/apachedft/conf/httpd.conf:
ScriptAlias /cgi-bin/ /QSYS.LIB/XMLSERVICE.LIB/
<Directory /QSYS.LIB/XMLSERVICE.LIB/>
  AllowOverride None
  order allow,deny
  allow from all
  SetHandler cgi-script
  Options +ExecCGI
</Directory>
```

```
stop/start Apache instance:
endTCPSVR SERVER(*HTTP) HTTPSVR(APACHEDFT)
STRTCPSVR SERVER(*HTTP) HTTPSVR(APACHEDFT)
```

Creating the HTML web page

Cut and paste the following HTML/XML (Example 5-19) form to your notebook Desktop/strsql.html file and complete the following steps:

1. Change the action target to your system action:
"http://myibmi/cgi-bin/xmlcgi.pgm"
2. Point your favorite browser at the HTML file:
file:///home/adcd/Desktop/strsql.html
3. Enter the database (*LOCAL), user (your profile), and password (your password). Enter a SQL query in the HTML **strsql** command and click **STRSQL**.

Example 5-19 Sample HTML page

```
desktop/strsql.html:
<html>
<head>
<script>
function getVal() {
xml = "<?xml version='1.0'?>";
xml += "<myscript>";
xml += "<sql>";
xml += "<query>";
xml += document.getElementById('strsql').value;
xml += "</query>";
xml += "<fetch block='all' desc='on'>";
xml += "</fetch>";
xml += "</sql>";
xml += "</myscript>";
document.getElementById('xmlin').value = xml;
}
</script>
</head>
<body>
```

```

<h3>STRSQL</h3>
<form onsubmit="getVal();" name="input"
action="http://myibmi/cgi-bin/xmlcgi.pgm" method="post">
<br><input type="text" name="db2" value="*LOCAL" size="40" > database
<br><input type="text" name="uid" value="MYUID" size="40" > user
<br><input type="password" name="pwd" value="MYPWD" size="40" > password
<input type="hidden" name="ipc" value="*NA">
<input type="hidden" name="ctl" value="*here *cdata">
<input type="hidden" name="xmlin" id="xmlin" value="na">
<input type="hidden" name="xmlout" value="500000">
<br><input type="text" name="strsql" id="strsql" size="40" /> strsql command
(select * from db2/animals)
</table>
<br><br><input type="submit" value="STRSQL" />
</form>
</body>
</html>

```

Desktop/strsql.html example

As the strsql.html name implies, this simple HTML file enables STRSQL to run from your notebook on IBM i. Enter any SQL statement that you want in the HTML form and XMLSERVICE runs just like an STRSQL green screen when you click **STRSQL**. The XMLSERVICE output that is returned is XML, so if your browser has issues displaying XML, you might have to view the page source.

The XMLSERVICE input is plain XML, which is used to make the request “select * from db2/animals”, as shown in Figure 5-49.

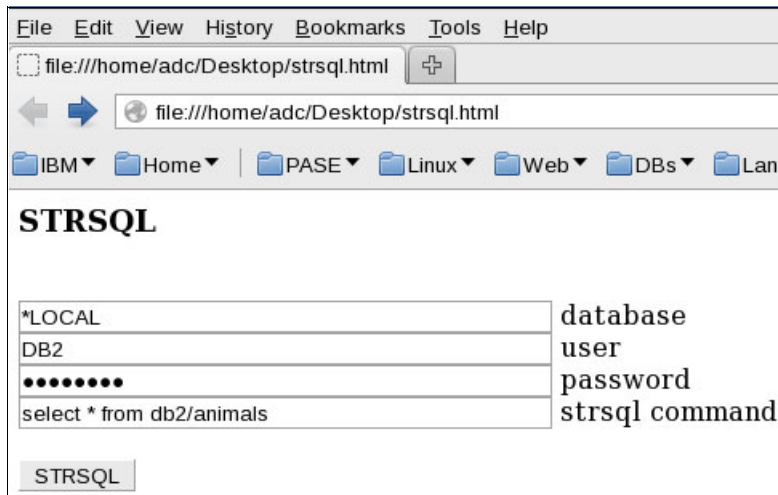


Figure 5-49 STRSQL HTML page for submitting a REST request

The HTML form strsql.html uses a simple JavaScript function `getVal()` with `document.getElementById('strsql').value`, which reads HTML text input `<input id='strsql'>` and adds the user SQL request to the XML document to HTML text input `<input id="xmlin">`. All XMLSERVICE required REST tag elements can be seen in the following HTML form:

```

http://myibmi/cgi-bin/xmlcgi.pgm?db2=*LOCAL@uid=MYUID@pwd=MYPWD@ipc=*NA@ctl="*here
*cdata"@xmlin=(see JavaScript)@xmlout=500000.

```

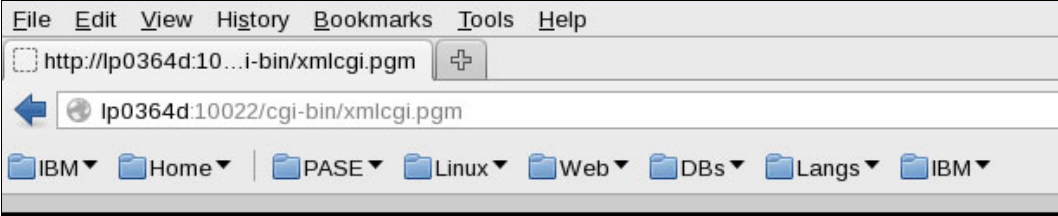
What is happening

If you change method="post" to method="get", the fully encoded document appears on the browser URL line. As you can see, when input arrives from our browser (or REST client), XMLSERVICE/XMLCGI.PGM has much HTTP decoding to do before parsing the XML document and servicing the request.

The flow

1. Point your browser to file:///home/adc/Desktop/strsql.html.
2. Enter the SQL query and click **STRSQL**.
3. IBM i Apache XMLCGI.PGM receives your encoded HTML/XML form action="http://myibmi/cgi-bin/xmlcgi.pgm".
4. XMLCGI.PGM calls XMLSERVICE.PGM (using the DB2 stored procedures iPLUGxxx).
5. XMLSERVICE.PGM parses the XML input and runs the internal DB2 driver <sql><query>...</query></sql>.
6. XMLSERVICE.PGM parses the result set from the DB2 driver into an output XML document <sql><fetch/></sql>.
7. The browser sees the XML return of the DB2 data.

The XMLSERVICE output is plain XML output from XMLSERVICE with records returned from db2/animals.



```
-<myscript>
- <sql>
  - <query conn="conn1" stmt="stmt1">
    <success>+++ success select * from db2/animals</success>
  </query>
  - <fetch block="all" desc="on" stmt="stmt1">
    - <row>
      <data desc="ID">0</data>
      <data desc="BREED">cat</data>
      <data desc="NAME">Pook</data>
      <data desc="WEIGHT">3.20</data>
    </row>
```

Figure 5-50 XMLSERVICE output

XMLSERVICE is open source, so you can examine the internals of XMLCGI.PGM if you want. You need to understand that XMLCGI.PGM decodes the HTML/XML document, passes the XML document request to XMLSERVICE.PGM (DB2 request example), and returns the XML output to the client (browser). If the REST client is not a browser, but a scripting language such as PHP or Ruby, the exact same sequence occurs, except additionally most languages offer an XML parser to parse output XML into variables or structures (PHP Toolkit or Ruby Toolkit).

Quick test functions HTML/XML

The XMLSERVICE HTML/XML technique can be used for nearly anything on IBM i CMD, PGM, SRVPGM, system APIs, PASE utilities, DB2, and so on. You can copy the `strsql.html` form, modify it, and try other XMLSERVICE functions `<myscript>other functions</myscript>`. The HTML/XML technique is useful for testing a potential XMLSERVICE program service without writing a line of code and clearly demonstrates elegant simplicity that is embodied by XMLSERVICE.

XMLSERVICE DB2 interface

A DB2 connection is not a web service, but many languages support high-speed DB2 local/remote requests, so XMLSERVICE includes a stored procedures interface (iPLUG4K - iPLUG15M). The nature of the DB2 stored procedures requires a size to be specified on in/out parameters. Therefore, the XMLSERVICE library includes various iPLUGxx sizes to fit your XML document data needs (4 K, 32 K, 65 K, 512 K, 1 M, 5 M, 10 M, and up to 15 M).

XMLSERVICE DB2 (Example 5-20) is much faster over a REST interface, so many language toolkits offer DB2 connection as the premier service.

Example 5-20 RPG DB2 calling XMLSERVICE

```
myIPC = '/tmp/thebears1';
myCtl = '*sbmjob';
// call XMLSERVICE/ZZCALL(...) using XML only
myXmlIn =
  '<?xml version="1.0" encoding="ISO-8859-1"?>'           + x'0D'
+ '<script>'                                             + x'0D'
+ '<pgm name="ZZCALL" lib="XMLSERVICE">'              + x'0D'
+ '<parm><data type="1a" v="1">Y</data></parm>'          + x'0D'
+ '<parm><data type="1a" v="2">Z</data></parm>'          + x'0D'
+ '<parm><data type="7p4" v="3">001.0001</data></parm>'   + x'0D'
+ '<parm><data type="12p2" v="4">0003.04</data></parm>'  + x'0D'
+ '<parm>'                                              + x'0D'
+ ' <ds>'                                              + x'0D'
+ ' <data type="1a" v="5">A</data>'                    + x'0D'
+ ' <data type="1a" v="6">B</data>'                    + x'0D'
+ ' <data type="7p4" v="7">005.0007</data>'            + x'0D'
+ ' <data type="12p2" v="8">0000000006.08</data>'      + x'0D'
+ ' </ds>'                                             + x'0D'
+ '</parm>'                                            + x'0D'
+ '</pgm>'                                             + x'0D'
+ '</script>'                                          + x'00';
myXmlOut = *BLANKS;
// make call to XMLSERVICE provided stored procedure(s)
// sizes from iPLUG4k to iPLUG15M (see crtsql xmlservice package)
Exec Sql call XMLSERVICE/iPLUG4K(:myIPC,:myCtl,:myXmlIn,:myXmlOut);
```

For examples, see the Young i Professionals website:

<http://youngiprofessionals.com/wiki/index.php/XMLSERVICE/RPGWebServices>

5.5.3 Data interchange

In today's world, it is almost impossible to discuss the topic of data interchange without talking about XML. Although, in some respects, XML has not replaced all other forms of EDI to the extent that many of its proponents anticipated, it has established itself as the foundation for many data interface functions, including many web services.

If you are not using XML processing in your applications today, you will be soon. So, it is important that you become familiar with the basics. Processing XML is an important task.

It is beyond the scope of this section to provide a detailed description of what XML is, how it came to be designed, and how it is coded, but you can find an excellent quick introduction at <http://www.sitepoint.com/really-good-introduction-xml/>. In addition to XML, this section provides a brief introduction to another data interchange language, JSON.

XML versus HTML

For the sake of this description, the main thing that you need to understand about XML is that its purpose is to describe both the content and the nature of the information that it represents. In this, it differs from other markup languages such as HTML and User Interface Manager ((UIM), the language in which IBM i menus are written). These languages are concerned with the visual representation of data, not with its meaning. Therefore, they are of little help in system to system communication.

To understand what this means, look at the simple HTML extract in Example 5-21. As a human, you probably can guess that "Phones R Us" is the name of a company and that "30033" is a postal code instead of an account balance. But there is nothing inherent in the extract that tells you this. For example, suppose that you wanted to write a program to enable you to answer questions such as which customers are based in the state of Georgia. If the HTML includes text column headings for the table, as opposed to using graphics as some web pages do, it is possible that you can write a program that does a reasonable job, but it is not an easy or reliable approach given the nature of HTML.

Example 5-21 Simple HTML document

```
...
<table>
  <tr>
    <td>938472</td>
    <td>Doe</td>
    <td>4859 Elm Ave</td>
    <td>Redbookstown</td>
    <td>TX</td>
    <td>75217</td>
  </tr>
  <tr>
    <td>593029</td>
    <td>Smith</td>
    ...
  </table>
```

Now look at the corresponding XML sample in Example 5-22. The document contains the same basic information as in the HTML example, but because XML allows you to create your own descriptive element names, it is much easier to process. Even without direct RPG support for XML processing, it is not hard to answer the question that was posed earlier.

Example 5-22 XML document

```
<Customers
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="Customers.xsd">
  <RecordCount>3</RecordCount>
  <Customer ID="938472">
    <Name>Doe</Name>
    <Address>
      <Street>4859 Elm Ave</Street>
      <City>Redbookstown</City>
      <State>TX</State>
      <Zip>75217</Zip>
    </Address>
  </Customer>
  <Customer ID="593029">
    <Name>Smith</Name>
    ...
</Customers>
```

As you will appreciate as you learn more about XML, this is an overly simplistic example but it serves our purpose. In practice, the element names and the required structure normally are determined by the provider of the document or the web service you are consuming.

One other point: The XML specifications incorporate the notion of schemes. Schemes provide a means of describing the rules for formatting the document, allowing its validity to be checked. The closest comparison, in conventional IBM i terms, is things such as the length, data type, and validation specifications in DDS. In Example 5-22, the XML Schema Definition (XSD) file `Customers.xsd` is referenced at the beginning of the document. A simple version of that schema is shown in Example 5-23 and described below (the numbers in parentheses correspond to the numbers in the explanation). We do not spend much time on this topic because most readers never need to design an XSD, but you should be able to recognize one, and even make minor changes, when you need to.

Example 5-23 Sample XML schema for the Customers document

- (1) `<?xml version="1.0" encoding="ISO-8859-1" ?>`
`<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">`
- (2) `<xs:element type="xs:int" name="RecordCount"/>`
`<xs:element type="xs:string" name="Name"/>`
`<xs:element type="xs:string" name="Street"/>`
`<xs:element type="xs:string" name="City"/>`
`<xs:element type="xs:string" name="State"/>`
`<xs:element type="xs:int" name="Zip"/>`
- (3) `<xs:attribute type="xs:int" name="ID"/>`
- (4) `<xs:element name="Address">`
`<xs:complexType>`
`<xs:sequence>`


```

        <xs:element ref="Street"/>
        <xs:element ref="City"/>
        <xs:element ref="State"/>
        <xs:element ref="Zip"/>
    </xs:sequence>
</xs:complexType>
</xs:element>

(5) <xs:element name="Customer" >
    <xs:complexType>
    <xs:sequence>
        <xs:element ref="Name"/>
        <xs:element ref="Address"/>
    </xs:sequence>
    <xs:attribute ref="ID"/>
    </xs:complexType>
</xs:element>

(6) <xs:element name="Customers">
    <xs:complexType>
    <xs:sequence>
        <xs:element ref="RecordCount"/>
        <xs:element ref="Customer" maxOccurs="unbounded"
            minOccurs="1"/>
    </xs:sequence>
    </xs:complexType>
</xs:element>

</xs:schema>

```

Here is a basic description of the schema:

1. This initial section is just identifying the version of XML that is being used and the fact that this is a schema.
2. There are multiple ways in which the schema can be coded. Here, use what we find to be the simplest approach, namely to first identify all of the individual building blocks (that is, the basic elements) and then to group them into the structures that form the document. RecordCount and Zip are defined as being integers. All other elements are identified as strings. If you have restrictions on the range of values that an element can contain, this is where you put them.
3. In addition to the basic elements, we also define the attribute ID.
4. We begin the process of defining the structure of the compound (or group) elements. The Address element is defined as being a complex type. We create our own data type that is called an Address and specifying its components and rules, in this case, the sequence of elements that it is composed of. You can think of this action as similar to defining a data structure (DS) in RPG. Indeed, as you see later, there is a direct correlation between compound elements and DS when processing XML documents with the RPG XML-INTO.
5. This schema defines the Customer element as including the Name and Address elements and associates the ID attribute with Customer.
6. We define the root element, Customers, as consisting of the RecordCount followed by a minimum of one Customer elements and an unlimited maximum.

We do not go into further detail here because schema support is not built into the RPG XML support. Later, we describe a technique for validation by using IBM i 7.1 SQL support for XML.

For more information about this topic, see the following website:

<http://www.w3schools.com/xml/>

Generating XML

This section describes how to generate XML in RPG programs. There are basically three ways to do this task:

1. Build the XML strings manually, that is, create the XML string by concatenating literals and field values.
2. Use tools to help the generation. There are two main types of tools in this area. The first tool uses an API approach to build the document one piece at a time. The second tool involves the usage of a template or mail merge type operation.
3. The third option is to use the latest built-in capabilities of DB2.

For our example, we consider only those tools that are built in to the operating system or the RPG compiler, or are available at no charge. There are other third-party tools that you might consider. Details about the third option, the XML capabilities of DB2, can be found in 5.5.4, “Database access” on page 163.

Building XML manually

As you can see in Example 5-24, this method is not difficult to understand. Indeed, for situations such as forming simple web service requests, it is a perfectly adequate way of building XML.

Always use a varying length field, as shown in Example 5-24, to build your XML string. You find that it is considerably more efficient than using a fixed-length field, which forces you to constantly trim the field as you add to its content. For more information about this topic, see Chapter 7, “Modern RPG” on page 187.

Here is a simple example of building the XML document that is shown in Example 5-23 on page 146. For the sake of simplicity, the portion where the document is written to the IFS or transmitted to the web service and so on, is omitted from the examples.

Example 5-24 Building XML data manually

```
D customer      E DS                      ExtName(QIWS/QCUSTCDT)
D xmlString     s                      10000a  Varying
D recordCount  s                      5i 0
D i            s                      5i 0
D endOfData    c                      '02000'

/Free
Exec SQL
  select count(*)
  into :recordCount
  from QIWS/QCUSTCDT;

xmlString = '<Customers><RecordCount>' +
           %Char(recordCount) +
```

```

        '</RecordCount>';

Exec SQL
  declare customerCursor cursor for
    select *
    from QIWS/QCUSTCDT;

Exec SQL
  open customerCursor;

DoU SQLSTATE = endOfData;
Exec SQL
  fetch next from customerCursor into :customer;

If SQLSTATE <> endOfData;
  xmlString += '<Customer ID="' + %editc(cusnum:'X') + '">+
    <Company>' + %TrimR(company) + '</Company>'+
    <Address>+
      <Street>' + %TrimR(street) + '</Street>'+
      <City>' + %TrimR(city) + '</City>'+
      <State>' + %TrimR(state) + '</State>'+
      <Zip>' + %EditC(zipcod:'X') + '</Zip>'+
    </Address>+
  </Customer>';

  EndIf;
EndDo;

Exec SQL
  close customerCursor;

// XML document is now complete in the variable xmlString
// and ready to be used.

```

You do not need to spend much time studying this code to see how bad things might get if you try to use this approach to create more complex documents. What if, in response to a change in the web service that we pass this data to, you had to change all the element names? Or put the elements in a different sequence? What if a second, optional, Street element is added? What if optional attributes such as Customer type must be added to the Customer element? Updating XML manually is not always the best choice.

Maintenance of code that includes literals can be a nightmare, particularly when some of the components are optional. Even if you substitute named constants for the literal strings, it is still easy to make mistakes, and hard to spot them when you do. The free-format D-specs in IBM i 7.1 TR7 make this task a little less error-prone because longer literals can be specified, but it is still problematic.

Tools for XML generation

This situation is where using tools really comes into play. At the time of writing, we are aware of three no-charge tools that are commonly used for generating XML:

- ▶ **CGIDEV2:** This tool is probably the most widely used because it has been around the longest. It was originally designed for the purpose of building web pages, but, because of the nature of XML, it is equally at home in that environment.
- ▶ **powerExt:** This tool is also primarily designed for building web applications (<http://powerext.com>) but more tools have been added to its CGIDEV2 base specifically for building and using XML.
- ▶ **XMLi:** This tool (<http://sourceforge.net/projects/xmli/>) is designed for the generation of XML and offers two distinct methods.

We have included here brief examples of producing the same basic XML document using each of these tools. We explain why you might want to use one over the other as we describe each tool.

Building XML with CGIDEV2

CGIDEV2 uses a template-based system to build XML. You can download it from the following website:

<http://www.easy400.net>

You can see the basic template that we use in our example in Example 5-25.

Here are the main points to note in the template:

- ▶ *Record formats* (known as sections in CGIDEV2 terms) are introduced by the characters “/\$”. So, this template contains three formats: Header (a), Customer (b), and Trailer (c).
- ▶ Within each section, there are a number of substitution variables. They begin with the characters “/” and end with “%”. This template contains the substitution variables count, ID, and name, among others. Think of these variables as operating in the same way as variable names within the DDS for a printer or display file. Their content is replaced at run time by the content of the associated variable.

Example 5-25 Template that is used by the CGIDEV2 program

```
(a) /$Header
    <Customers>
      <RecordCount> /%count%/ </RecordCount>
(b) /$Customer
    <Customer ID="/%id%/">
      <Name>/%name%/</Name>
      <Address>
        <Street>/%street%/</Street>
        <City>/%city%/</City>
        <State>/%state%/</State>
        <Zip>/%zip%/</Zip>
      </Address>
    </Customer>
(c) /$Trailer
    </Customers>
```

The CGIDEV2 program that uses this template is shown in Example 5-26. Here are the three basic phases that are involved in the process:

- ▶ Load the template (Example 5-25 on page 150) in to the program through a call to the **getHTML** function. This function identifies the name of the template to load and from where it is to be loaded. This name applies to our example. You can select your own name when you develop your application. CGIDEV2 was originally designed to generate HTML, so do not be concerned by the usage of that term in the function names.
- ▶ Set the content of the substitution variables (the data values) by using the **updateHTMLvar** function.
- ▶ Write the buffered XML document out to an IFS file. Alternatively, if you simply need to access the data content directly to pass to a web service, you can use CGIDEV2 to access the buffer directly.

Example 5-26 CGIDEV2 program

```

h DftActGrp(*No) Option(*SrcStmt : *NoDebugIO)
  h BndDir('CGIDEV2/TEMPLATE2')

      // Copy in standard CGIDEV2 prototypes
      /copy CGIDEV2/qrpglesrc,prototypeb
      /copy CGIDEV2/qrpglesrc,usec

D templateName      s              128a  Varying
D                                                            Inz('/Redbook+
D                                                            /Customers_T.xml ')

D xmlFilename       s              128a  Varying
D                                                            Inz('/Redbook+
D                                                            /Customers2.xml ')

D customer          E DS              ExtName('QIWS/QCUSTCDT')

D recordCount       s                5i 0
D endOfData         c                '02000'
D clearVars         c                '0'

/Free

(A) getHTMLIFS(templateName);

      Exec SQL
      select count(*)
      into :recordCount
      from QIWS.QCUSTCDT;

(B) updHTMLVar('count': %Char(recordCount): clearVars );

      wrtSection('Header');

      Exec SQL
      declare customerCursor cursor for
      select CUSNUM, LSTNAM, INIT, STREET, CITY, STATE, ZIPCOD
      from QIWS.QCUSTCDT;

      Exec SQL

```

```

        open customerCursor;

DoU SQLSTATE = endOfData;
  Exec SQL
    fetch next from customerCursor into :customer;

  If SQLSTATE <> endOfData;
    updHTMLVar('id': %EditC(CUSNUM: 'X') );
    updHTMLVar('name': (%TrimR(INIT) + ' '
      + %TrimR(LSTNAM)) );
    updHTMLVar('street': %TrimR(STREET) );
    updHTMLVar('city': %TrimR(CITY) );
    updHTMLVar('state': STATE );
    updHTMLVar('zip': %EditC(ZIPCOD: 'X') );

    wrtSection('Customer');
  EndIf;

EndDo;

  Exec SQL
    close customerCursor;

  wrtSection('Trailer');
(C) wrtHtmlToStmf(xmlFilename: 1208);

  *InLR = *On;

/End-Free

```

Although CGIDEV2 provides an easy-to-use mechanism, it does have some limitations:

- ▶ It does not automatically encode any reserved characters that are encountered in the data. So, if any characters that have special meaning in HTML and XML, such as “&”, “<”, and “>”, are included in substitution variables, they create problems. CGIDEV2 does provide the **encode()** function to handle this situation, but it was designed for HTML and deals only with those characters that need special encoding in HTML. You must customize the routine to have it handle the additional characters in XML.
- ▶ CGIDEV2 becomes difficult to use if the XML contains many optional elements. In this case, you must make a choice between using large numbers of small template units and building optional portions of the XML manually. In these cases, it can be simpler to use an approach where individual elements are added to the XML stream one piece at a time. This is the approach that is used by the powerExt and XMLi APIs.

For a more detailed explanation of using CGIDEV2 for XML generation, go to the following website:

<http://www.ibmssystemsmag.com/ibmi/developer/rpg/Using-CGIDEV2-for-Generating-XML>

Building XML with powerExt

The approach that is taken by powerExt is to use individual API calls to add nodes and attributes to the XML tree. You can download powerExt from the following website:

<http://www.powerExt.com>

As you can see in Example 5-27, a call to the API `xmlNode()` is used to add a node regardless of whether it is an elementary item (B) or a group item (A). The only difference is whether the element value itself is present.

The API `XMLendnode()` is used to close off open elements (C). There is no need to specify the name of the element that is closing because the rules for formulating XML make it possible for the tool to work out for itself the element name to use. That is, the last one that was opened, but not yet closed, must be the one to be closed.

Example 5-27 powerExt program

```

/copy pextcd2/qsrc,pxapihdr

    // powerEXT API Connectors
    /copy pextcd2/qsrc,pxapicgicn

    // Declare File For SQL
    d customer      E DS              ExtName('QIWS/QCUSTCDT')

    * Declare Internal Variables
    d recordCount   s                5i 0
    d endOfData     c                '02000'
    d xmlFilename   s                128a Varying
    d               Inz('/Redbook+
    d               /Customers.xml')

    /free

    clearSrvPgm();
    setContent('*none');

    Exec SQL
        select count(*)
            into :recordCount
            from QIWS.QCUSTCDT;

    (A) xmlNode('Customers');
    (B)  xmlNode('RecordCount':'':%char(recordCount));

    Exec SQL
        declare customerCursor cursor for
        select * from QIWS.QCUSTCDT;

    Exec SQL
        open customerCursor;

    DoU SQLSTATE = endOfData;
        Exec SQL
            fetch next from customerCursor into :customer;

    If SQLSTATE <> endOfData;

        xmlNode('Customer':'ID="'+ %char(cusnum) + "'');
        xmlNode('Name':'':LSTNAM);
        xmlNode('Address');
        xmlNode('Street':'':street);

```

```

        xmlNode('City':'':city);
        xmlNode('State':'':state);
        xmlNode('Zip':'':%editc(zipcod:'X'));
        xmlEndNode();
    xmlEndNode();

    EndIf;
EndDo;
Exec SQL
    close customerCursor;

(C) xmlEndNode();

    echoToStmf(xmlFilename:1208);

    *inlr = *on;
/end-free

```

Building XML with XMLi

XMLi offers not one but two different modes of operation. You can download XMLi at the following website:

<http://sourceforge.net/projects/xmli/>

The first is an API approach that is similar to the one that is used by powerExt. The principal difference is that, with XMLi, when closing an element, it is necessary to specify the element name. Because it is otherwise so similar, we have not included an example here, but you can find many examples of its usage in the downloaded package.

The second option that is offered by XMLi appears at first glance to be similar to the template approach of CGIDEV2, and in some respects that is a valid comparison. However, XMLi templates go far beyond the capabilities that are offered by CGIDEV2.

XMLi templates can incorporate SQL so that the entire data retrieval and XML build process can be simplified (Example 5-28). In this simple example, you can see the relevant SQL queries that are embedded in the template (A). We have extended the SQL in this case to incorporate a WHEN clause.

If you are familiar with XML transforms, the syntax of the XMLi template should be familiar to you. For a more detailed explanation, go to the following website:

<http://www.ibmssystemsmag.com/ibmi/developer/rpg/xmli>

Example 5-28 Sample XMLi template

```

<xmli:template xmlns:xmli="http://www.sourceforge.net/xmli"
               ccsid="1208" format="pretty">
  <Customers>
    <xmli:run-sql name="custCount"
(A)      statement="select count(*) from QIWS.QCUSTCDT
                  where STATE = 'TX'">
    </xmli:run-sql>
    <xmli:for-each>
      <RecordCount><xmli:value-of select="custCount.1" />
      </RecordCount>
    </xmli:for-each>

```



```

    <xmli:run-sql name="custRow"
      statement=
(A)      "select CUSNUM, LSTNAM, STREET, CITY, STATE, ZIPCOD
          from QIWS.QCUSTCDT where STATE = 'TX'">
    </xmli:run-sql>
    <xmli:for-each>
      <Customer ID="{custRow.1}" >
      <Name><xmli:value-of select="custRow.2" /></Name>
      <Address>
      <Street><xmli:value-of select="custRow.3" /></Street>
      <City><xmli:value-of select="custRow.4" /></City>
      <State><xmli:value-of select="custRow.5" /></State>
      <Zip><xmli:value-of select="custRow.6" /></Zip>
      </Address>
      </Customer>
    </xmli:for-each>

</Customers>

<xmli:write-to-file path="'/Redbook/CustomersE2.xml'" />

</xmli:template>

```

You see that some aspects of the template look remarkably similar to the CGIDEV2 version. However, the code that populates the template (Example 5-29) is different and mostly remarkable for how little code is needed. Indeed, in this example, we might have avoided writing an RPG program altogether and instead used the CL commands that are supplied with the package to drive the execution of the template.

Example 5-29 Sample XMLi program

```

/include XMLILIB/QRPGLESRC,XMLI_H

D templateFilename...
D          s              128a  Varying
D                               Inz('/Redbook/XMLi+
D                               Template1.xml')

D err          s              10i  0

/Free
err = xmli_loadConfig('/xmli/config/xmli_config.xml');
// Load and run the template...
xmli_loadTemplate('CUST' : templateFilename);
err = xmli_runTemplate('CUST');

// Optionally unload template (or leave loaded for reuse)
xmli_unloadTemplate('CUST');

*InLR = *On;

```

As you can see, XMLi offers some powerful capabilities and can be used by anyone with a basic knowledge of SQL and XML.

Consuming XML with RPG

In Version 5 Release 4, RPG added direct XML support to the language in the form of two new opcodes and their associated Built In Functions (BIFs).

The first and most commonly used of these opcodes is XML-INTO. It works by parsing the XML document and matching the element names and hierarchy with the names in a corresponding data structure. To use this support, you must be familiar with the usage of nested data structures. Even the simple XML document cannot be handled by a simple DS. It is necessary to use a nested data structure to represent the XML document hierarchy.

The second of the new opcodes is XML-SAX. It provides a low level “one-piece-at-a-time” method of processing an XML document. It is effectively the technology on which XML-INTO itself is based. We do not include an example in this publication. It is sufficient to say that it works by identifying each individual component (element name, element data, attribute name, and so on) that it encounters in the XML document and hands control to a user-defined subprocedure. If you want to read more about this topic, go to the following websites:

- ▶ <http://www.mcpressonline.com/programming/rpg/rpg-has-sax-appeal.html>
- ▶ <http://iprodeveloper.com/rpg-programming/rpgs-xml-sax-opcode>

(In Version 6 and later releases, the problems with XML-INTO that are described in these articles were corrected by IBM.)

Processing XML with XML-INTO

There is one feature of RPG IV that you must understand if you are to process XML documents with the RPG built-in support, which is *nested data structures*. This is a feature of the language that was partly introduced in Version 5 Release 1 and enhanced in Version 5 Release 2. Version 6 rounded out the support by adding the TEMPLATE keyword, which we have used in our example. It is beyond the scope of this publication to teach that topic here, but there are many articles that you can read on the web:

- ▶ <http://www.ibmssystemsmag.com/ibmi/developer/rpg/iSeries-EXTRA--Data-Structure-Enhancements-in-V5R1/>
- ▶ <http://www.itjungle.com/fhg/fhg091609-story02.html>

Here, we limit our description to the usage of these features as they pertain to our example.

Table 5-3 on page 157 shows an extract from the DS that is used in our sample program together with the corresponding XML elements in the document. We have also indicated the essential keywords used (Dim, LikeDS, and Template).

Table 5-3 D-Spec definition for corresponding XML example

D-Spec definitions	Corresponding XML elements
d customers ds	<Customers>
d recordCount	<RecordCount>
d cnt_customer	<Customer
d customer (LikeDs)	
d (Dim)	
d customer_T ds (Template)	ID="P0230A">
d id	<Name>
d name	<Address>
d address (LikeDS)	
d address_T ds (Template)	<Street>
d street	<City>
d city	<State>
d state	<Zip>
d zip	

Now that you have seen the basic correlation between the RPG structures and the XML, we look at the RPG example in more detail (Example 5-30). There is a number in parenthesis on the far left side. That number corresponds to the following detailed description.

Example 5-30 Detailed RPG example

```

(1) d customers ds Qualified
(2) d recordCount 5i 0
(3) d cnt_customer 5i 0
(4) d customer LikeDS(customer_T)
    d Dim(9999)

(5) d customer_T ds Template Qualified
(6) d id 6a
    d name 8a
(7) d address LikeDS(address_T)

(8) d address_T ds Template
    d street 13a
    d city 6a
    d state 2a
    d zip 5a

(9) d xmlFilename s 128a Varying
    d Inz('/Redbook+
    d /Customers.xml')

(10) XML-INTO customers %XML( xmlFilename:
(11) 'doc=file
(12) case=any
(13) countprefix=cnt_' );

(14) If customers.cnt_customer <> customers.recordCount;
```

```

    Dsply 'Count mismatch';
Else;
    Dsply 'Counts matched';
EndIf;

```

1. The DS Customers is the target of the XML-INTO operation (N). After the operation, the DS contains all of the data from the entire XML document. The keyword QUALIFIED is required any time that you must use the LIKEDS keyword to nest another data structure (D) within the one that you are defining.
2. This item defines the field to contain the <RecordCount> element. In this example, we have defined it as numeric to demonstrate that RPG can handle the character to numeric translation for you. However, you should not use this feature unless you are validating the XML before processing, or know for certain that the element always contains a valid numeric value. The reason for this is that the XML-INTO operation causes an error if a non-numeric value is encountered and the program is not aware of exactly why. For this reason, it is normally recommended that you define such fields as character and then validate and translate them yourself.
3. The cnt_customer field is completed by RPG with a count of the number of customer elements that are encountered. We describe this task in more detail when we look at the usage of the countprefix option (M) in step 13 on page 159.
4. Because the customer element in the XML is a repeating compound element, it is defined by referencing the customer_T template DS and adding a DIM to define the maximum number of repeats.
5. Define the content of the customer_T template. The actual sequence of fields within the DS is not important. The only requirement is that they be spelled the same as in the XML document, and be in the same position in the hierarchy. By this, we mean that city can follow state in the address DS, but it cannot be placed directly in the customer DS.
6. Notice that the customer attribute ID is placed in the customer DS. Attributes are treated as being at the same hierarchical level as children of the element that they belong to. For example, ID is at the same level as name.
7. Address is also a compound element and is defined by referencing the address_T template.
8. The last component is the address_T template, which defines the component address fields.

If you are not yet familiar with the notion of nested DS, you might be wondering how the data in this structure is referenced. It is simple: Take the Customer ID for the first customer in the DS. It is referenced as `customers.customer(1).id`. The street for this same customer is `customers.customer(1).address.street`. It might look a little strange until you get used to it, but it is self-explanatory.

Having looked at the details of the target DS, consider the rest of the code.

9. This is the definition for the XML file name. Using a variable length field for IFS names is usually a good idea because it prevents having to worry about trimming trailing spaces from the name.
10. The XML-INTO operation specifies customers as the target. The source of the XML document is identified by the %XML built-in function through its first parameter (`xmlfilename`).
11. This option tells the compiler that the first parameter identifies the name of the file containing the XML. Without this option, the compiler assumes that the XML document itself is contained within the parameter.

12. You usually need this option. It tells the compiler to convert all element and attribute names to uppercase before comparing them with the RPG names.
13. The countprefix option identifies the naming convention to be used for any element for which you want RPG to supply a count of the number of instances of that element that it encounters. In this case, we need to count the number of <customer> elements. We do this by naming the field cnt_customer and placing it in the DS at the same hierarchical level as the customer element. Why ask RPG for this count rather than rely on the <RecordCount> element value? For one thing, it helps us validate the supplied count, but more importantly, if fewer <customer> elements are found than specified in the DIM, then the RPG default behavior is to issue an error while parsing the document. By requesting the count, we are telling the compiler that we are aware of this possibility and do not need it to issue an error. This option is useful when handling optional elements because it provides a simple way of detecting whether they are present in the XML.
14. Having extracted all of the data, we had to do something with it. We chose to simply compare the supplied count with the one calculated by RPG.

As you have probably determined by now, the hardest part of using XML-INTO is not coding the operation itself, but instead coding the data structures that are needed to contain the data. There are many other aspects to using XML-INTO that you must learn to fully use its power, but they are beyond the scope of this brief introduction. For example:

- ▶ Because it is vital that the names match, in many cases RPG support for long names is needed.
- ▶ Element names might contain characters that are invalid in RPG names. The %XML option case=convert handles this task.
- ▶ XML documents use of namespaces. There are many %XML options that relate to this task. Details can be found in the RPG Cafe at the following website:

<http://ibm.co/1d4Ux1z>

Example 5-31 is another example of how to use the XML-INTO built-in function to parse an XML document into a usable RPG Data Structure.

The first step in doing this task is to define the RPG Data Structure that matches the XML document elements. This is the target of the parsed data. Each subfield name in this data structure must match an element or attribute name in the XML document. If there is no corresponding field name, then the element or attribute goes unparsed.

The data structure for the Order root element is shown in Example 5-31.

Example 5-31 RPG Data Structure for XML parsing

```
//*****
// Local Data Structures
//*****
d DetailsDs      ds              qualified
d  numLine      10i 0
d  line         likeds(ZRDITEM) Dim(10)
d
d Order         ds              qualified
d  number       10i 0
d  customer     likeds(ZRDCUSTOMER)
d  details      likeds(DetailsDs)

*****
* SQL Table and Data structures
```

```

*****
d ZRDITEM      E ds          extname(OrderItem) qualified
d              d            Template alias
d number      e            EXTFLD(itemRec)
d item        e            EXTFLD(itemNumber)
d orderqty    e            EXTFLD(qty)
d
d ZRDCUSTOMER E ds          extname(Customer) qualified
d              d            Template alias
d number      e            EXTFLD(CustNumber)
d name        e            EXTFLD(Name)
d addr1       e            EXTFLD(address)
d *city       e            EXTFLD(city)
d *state      e            EXTFLD(state)
d zip         e            EXTFLD(zipcode)

```

In the Order data structure, the “number” subfield is named as the Order “number” attribute that is defined in the XML document. The “customer” subfield data structure is named as the “customer” child element. In the Order data structure, the “number” subfield is named as the Order “number” attribute that is defined in the XML document. The “customer” subfield data structure is named as the “customer” child element in the XML document. The “details” subfield data structure is named as the “details” child element in the XML document.

To support lists such as in the “details” child element, a subfield array is used for each array element. An additional subfield can be used to set the number of array elements returned. In this case, the DetailsDs.numLine subfield. This option is controlled by the “countprefix=num” XML option that is supplied to the %XML built-in function. When the parser finds a list such as the “line” elements, it returns the total number in the list to the subfield named with the prefix+element Name. In this case, it is “numLine”.

Finally, to map the XML element or attribute names to existing subfields, such as the ZRDITEM externally described data structure, the non-matching fields must be renamed to match the element names. For example, the “itemRec” field name is remapped using **EXTFLD()** to match the “number” attribute in the XML document.

In the call to the XML-INTO, the XML options are set using the %XML built-in function, %XML is used as the second operand of the XML-SAX and XML-INTO operation codes to specify the XML document to be parsed, and the options to control how the document is parsed. In this example, the options are set to read from an external file(doc=file) (Example 5-32).

Example 5-32 RPG code using XML-INTO to parse the XML file

```

xmlfile = 'Order.xml';
// case=any is needed to allow the case to be mixed
// allowextra=no is needed to allow for extra subfields
// allowmissing=yes is needed to allow for < number of
//           array elements (dim(10). XML has 3.
// ccsid=best ifs file CCSID is 1252... the file will get converted
//   to ucs-2 first then processed, and converted back to job ccsid
// countprefix=num is needed to set the count of the items in order
//   Order.details.numLine
xmlopt = 'doc=file ' +
        'allowextra=yes ' +
        'allowmissing=yes ' +
        'case=any ' +
        'countprefix=num ' +

```

```
'ccsid=best';  
XML-INTO Order %XML(xmlfile : xmlopt);
```

After the call to XML-INTO, the Order data structure is populated with the XML document data.

This example is a basic example of doing XML parsing in a native manner from RPG.

Alternative options

If for you do not like, or for some reason cannot use, the built-in RPG support for using XML, then there are many other options that are available to you:

- ▶ DB2: DB2 has been increasing its support for XML over a number of releases and is improving all the time. The latest additions allow for the easy transformation of XML documents to DB2 tables. You can find many of the details at the following websites:

- http://www.ibmssystemsmag.com/ibmi/developer/general/xml_db2_part1/
- http://www.ibmssystemsmag.com/ibmi/developer/general/xml_db2_part2/

Also, see 5.5.4, “Database access” on page 163.

- ▶ eXpat: Users of Scott Klement's HTTPAPI are familiar with the open source package eXpat (<http://expat.sourceforge.net>), which Scott has ported to the IBM i. It is available from his website as part of the HTTPAPI package or as a separate download from <http://www.scottklement.com/expat/>.

In its basic operation, eXpat is a SAX parser and is similar in operation to the RPG XML-SAX, albeit a little more versatile. However, it works only in this mode and does not automate the unloading of XML data into RPG DS in the same way as XML-INTO.

- ▶ powerExt: In addition to the XML generation capabilities described earlier, powerExt Core also incorporates the ability to process XML. It operates much like XML-SAX and eXpat with one significant difference. In both of those cases, the tool identifies all events and passes each one to your code for processing. The powerExt tool can do the same thing, but it is more commonly used to pass only the data for the elements and attributes that you specifically request to your program. This makes it a lot simpler to code for cases where you are only interested in specific values within the XML.
- ▶ Java and PHP options: There are many Java and PHP-based offerings in the open source world. If you are searching for specific capabilities in your XML processing routines, do not limit yourself to RPG-oriented options.
- ▶ There are many commercial products available from ISVs.

Validating XML

At the time of writing, the only XML validation capabilities that are built in to the system are those that are supplied with the latest releases of DB2. You can find examples of using DB2 in this manner at the following website, which also describes how to retrieve the error information in the event of a validation error:

http://www.ibmssystemsmag.com/ibmi/developer/rpg/unexpected_errors/

There are also many other options that are available. For example, if you have PHP installed on your system, then using its built-in XML validator is a relatively easy task that does not require a significant amount of knowledge of PHP. Similarly, you can use the standard XML validation routines that are found in the Java libraries to perform the same task.

JSON

XML is the choice for universal data exchange. However, with the huge uptick in web user interfaces and the usage of JavaScript to make them dynamic, JSON has become a preferred way to communicate data between web browsers and server-side applications. Like XML, it is language- and platform-independent. JSON offers advantages over XML when used in JavaScript:

- ▶ The JSON format is identical to the code for creating objects in JavaScript, which allows JavaScript to simply use the `eval()` opcode to run the JSON instead of parsing the text.
- ▶ JSON is not as verbose as XML. This reduced size is advantageous when transmitting data.
- ▶ JSON has a limited scope. XML was created to be a universal standard that is extensible and can be adapted to any situation. This makes using and parsing XML much more complicated. JSON is better for data exchange. XML was created for document storage and exchange.

JSON syntax can be described with a few simple rules:

- ▶ The data is in name-value pairs.
- ▶ The data is comma-separated.
- ▶ Square brackets are used to denote arrays.
- ▶ Curly braces are used to hold objects.

Our XML example can be represented as easily in JSON, as shown in Example 5-33.

Example 5-33 Sample JSON document

```
{
  "order": {
    "number": "987654",
    "customer": {
      "name": "Example Company",
      "addr1": "123 Main Street",
      "city": "Rochester",
      "state": "MN",
      "zip": "55901"
    },
    "details": {
      "line": [
        {
          "number": "1",
          "item": "123456",
          "unitprice": "$1.25",
          "orderqty": "300"
        },
        {
          "number": "2",
          "item": "147258",
          "unitprice": "$1.19",
          "orderqty": "100"
        },
        {
          "number": "3",
          "item": "963258",
          "unitprice": "$2.05",
          "orderqty": "25"
        }
      ]
    }
  }
}
```



```
}
  ]
}
}
```

As you can see, all of the data from our XML document can be represented easily in JSON, and our character count dropped from 490 to 363. Although that might seem insignificant, when dealing with real world data, that reduction in size can be significant.

Summary

If you must generate XML files and are already familiar with CGIDEV2, then the transition to using this tool to generate XML is a simple one. If you are not familiar with CGIDEV2 and do not intend to use it for web development, one of the other tools probably is a better choice for you.

One factor that might make you select powerExt is that it is as equally suitable for building JSON strings as it is XML. If you need to build both, it is a good choice.

If you must produce many different XML files with minimal programming effort, the XMLi powerful, template-based system makes it an excellent choice. This brief introduction includes only a small part of its features.

5.5.4 Database access

DB2 for i provides a wide variety of interfaces that allow data to be accessed from several programming languages and environments. When you develop applications on the server, DB2 for i can be accessed by using embedded SQL, Call Level Interface, and Java Database Connectivity (JDBC). When you use a remote client, DB2 for i can be accessed by using JDBC, Open Database Connectivity (ODBC), OLE DB, IBM Net.Data®, or other solutions that use IBM DRDA®. Through these interfaces, you can access DB2 for i in many programming languages using industry standards for database access.

Accessing DB2 for i from server applications

When you create an application that runs on IBM i, there are many programming languages that can be used. From within your programs, these applications might need to access data that is stored in DB2 for i. For most ILE languages, you can use the embedded SQL and Call Level Interface interfaces. For Java applications, DB2 for i is accessed using JDBC.

Embedded SQL

DB2 for i can be accessed by using Structured Query Language (SQL) statements that are embedded in programs that are created on the IBM i. IBM i supports the usage of embedded SQL for the C, C++, COBOL, PL/I, RPG, and REXX languages. When you use embedded SQL, SQL statements are placed directly in the source code. Information that is retrieved from DB2 for i is then placed in variables for processing by the program. More information about using embedded SQL can be found in the *Embedded Sql Programming* manual at the following website:

<http://pic.dhe.ibm.com/infocenter/iserics/v7r1m0/topic/rzajp/rzajpkickoff.htm>

Embedded SQL is useful when the SQL statements to be run are known at compile time. It also has a performance benefit over other interfaces because the SQL statements are prepared at compile time, as opposed to run time. The disadvantage of embedded SQL is that programs might not readily port to other platforms or environments.

Example 5-34 shows an example of a C program that uses SQL to run a query to obtain information. This particular query returns the name of the current user. When using C, the SQL statements are denoted by using the **EXEC SQL** keywords. When using embedded SQL, an **SQLCA** is used to return error information that is made available by using the **INCLUDE SQLCA** embedded SQL statement. An SQL cursor is then declared by using a query. The **OPEN C1** statement then opens the cursor and the **FETCH** statement is used to fetch the data in the variable *currentServer*. This variable is also known as a host variable. The **CLOSE** statement is used to close the cursor and then the C **printf** routine is used to print the retrieved value.

Example 5-34 Embedded SQL example using C

```
#include <stdio.h>

EXEC SQL INCLUDE SQLCA;

int main (int argc, char *argv[]) {
    char currentServer[20];

    EXEC SQL DECLARE C1 CURSOR FOR
        SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;

    EXEC SQL OPEN C1;

    EXEC SQL FETCH C1 INTO :currentServer ;

    EXEC SQL CLOSE C1;

    printf("Current system is %s.\n", currentServer);

}
```

The program is compiled by using the **SQLSQLCI** program. In this case, the source was in IFS and the module was compiled using **CRTSQLCI OBJ(SQLQUERY) SRCSTMF('/home/source/sqlquery.sqlc')** and the program was built using **CRTPGM SQLQUERY**. The program was then run using **CALL SQLQUERY**. The output of the program is shown in Example 5-35.

Example 5-35 Embedded SQL example output

```
Current System is MYSYSTEM.
Press ENTER to end terminal session.
```

Call Level Interface

Another way to access DB2 for i is by using the Call Level Interface. Call Level Interface is an interface that can be used by many programming languages and is an alternative to using embedded SQL. On the IBM i, the Call Level Interface is available to any of the Integrated Language Environment (ILE) languages. DB2 for i Call Level Interface also provides full Level 1 Microsoft ODBC support, plus many Level 2 functions. For more information about Call Level Interface, see the Call Level Interface reference manual, found at the following website:

<http://pic.dhe.ibm.com/infocenter/iseriess/v7r1m0/topic/cli/rzadpkickoff.htm>

A C program using Call Level Interface to accomplish the same work as the embedded SQL program is shown in Example 5-36. In the Call Level Interface program, the program must explicitly manage environment (SQLENV), connection (SQLHDBC), and statement (SQLHSTMT) handles. A connection to the local database is established by using the call to SQLConnect, specifying the “*LOCAL” database. When the connection is established, a statement handle is allocated and the query is run by using SQL Exec Direct. The Call Level Interface function SQL Bind Col is then used to bind the first column of the resulting row to the *currentServer* variable. The SQLFetch call then fetches the first row of data into the bound variable. After the data is fetched, the resources that are used by the program are released.

Example 5-36 Call Level Interface example using C

```

#include <stdio.h>
#include <sqlcli.h>

int main (int argc, char *argv[]) {
    char currentServer[20];
    SQLINTEGER currentServerLength;

    SQLHENV henv;
    SQLHDBC hdbc;
    SQLHSTMT hstmt;

    SQLAllocEnv (&henv);
    SQLAllocConnect (henv, &hdbc);

    SQLConnect (hdbc, "*LOCAL", SQL_NTS, NULL, SQL_NTS, NULL, SQL_NTS);
    SQLAllocStmt (hdbc, &hstmt);
    SQLExecDirect (hstmt, "SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1", SQL_NTS);
    SQLBindCol (hstmt, 1, SQL_C_CHAR, currentServer, sizeof(currentServer),
&currentServerLength);
    SQLFetch (hstmt);

    SQLCloseCursor(hstmt);
    SQLFreeStmt(hstmt, SQL_DROP);
    SQLDisconnect(hdbc);
    SQLFreeConnect(hdbc);
    SQLFreeEnv(henv);

    printf("Current user is %s.\n", currentServer);

}

```

This program was created by placing the source code in the IFS file /home/source/sqlquery.c. The module is compiled by using **CRTCMOD MODULE(CLIQUERY) SRCSTMF('/home/source/sqlquery.c')** and the program is created by using **CRTPGM CLIQUERY**.

Call Level Interface can also be used from the Portable Application Solutions Environment (PASE). A program that is written for PASE can easily use Call Level Interface to access DB2 for i. More information about PASE is found in the *Programming IBM PASE for i* manual, found at the following website:

<http://pic.dhe.ibm.com/infocenter/iseriess/v7r1m0/topic/rzalf/rzalfpase.htm>

Inside that manual, information about using Call Level Interface can be found in the section on Database access, found at the following website:

<http://pic.dhe.ibm.com/infocenter/iseriess/v7r1m0/topic/rzalf/rzalfdatabase.htm>

The same program that is shown in Example 5-36 on page 165 was compiled by using an IBM AIX® compiler. The resulting program can then be run within a shell, as shown in Example 5-37.

Example 5-37 Call Level Interface example output

```
$ ./sqlquery
Current system is <MYSYSTEM.
```

Because this application was using Call Level Interface, it can easily be ported to the PASE environment. In this case, the trivial port only required recompilation of the program.

PHP ibm_db2 driver

PHP is a scripting language that is also available on IBM i. From PHP, it is simple to access your IBM i database. From PHP, DB2 for i can be accessed by using the `ibm_db2` driver. More information about PhP on IBM i can be found at the following website:

<http://www.ibm.com/systems/power/software/i/php/>

A simple PHP program to access the database and run our sample query is shown in Example 5-38. In this example, we obtain the database, user, and password from the HTTP request. We then establish a connection by using the supplied database name, user name, and password. Then, the `db2_exec` function is used to run the select statement. Then, `db2_fetch_array` is used to retrieve the data for each row. The data is then formatted by using HTML.

Example 5-38 Database access with ibm_db2 driver from PHP

```
<?php

$database=$_GET['database'];
$user=$_GET['user'];
$password=$_GET['password'];

$conn = db2_connect($database,$user,$password);

$statement = "select current server from sysibm.sysdummy1";
$r = db2_exec($conn, $statement);

echo "<br>$statement\n";
echo "<table border='1'>";
while ($row = db2_fetch_array($r)) {
    echo ("<tr><td>".implode("</td><td>", $row)."</td></tr>");
}
echo "</table\n>";

?>
```

The output looks like Figure 5-51 on page 167.

```
select current system from sysibm.sysdummy1
MYSYSTEM
```

Figure 5-51 PHP example output

Java Database Connectivity

When you are creating applications that use Java, the JDBC API, included in the Java platform, enables Java programs to connect to a wide range of databases. Using JDBC, a Java program can issue SQL statements and retrieve data from DB2 for i. JDBC relies on JDBC drivers to connect to particular databases. IBM i provides two JDBC drivers to access the IBM i: the IBM Toolbox for IBM i JDBC driver and the native JDBC driver. The IBM Toolbox for IBM i JDBC driver is designed for used by client applications and is described in “Java and JDBC” on page 168. The native JDBC driver is a driver that is usable only by JDBC programs that are running on the IBM i. It is optimized to quickly transfer information between the database and Java. Information about the native JDBC driver is found in the IBM Developer Kit for Java manual, in the section on JDBC, found at the following website:

<http://pic.dhe.ibm.com/infocenter/iseriess/v7r1m0/topic/rzaha/rzahajdbcdriver.htm>

A Java program to accomplish the same work as the previous Call Level Interface example is shown in Example 5-39. In JDBC, the `DriverManager.getConnection` method is used to obtain a connection to a database. This method accepts one parameter, the JDBC URL. The JDBC URL specifies the JDBC driver to use, followed by the database to access and optional connection properties. The native JDBC driver is specified by including `db2` after the initial `"jdbc:"` value. The next part of the URL specifies the database to connect to. In this case, we specify the local database by using `"*local"`. After obtaining a connection, a `Statement` object is created from the connection. The `Statement` object is then used to run the query. Running a query produces a `ResultSet` object. The `rs.next()` method instructs the result set to move the next result, which is the first time it moves to the first row of the result. After the result set is positioned on a row, the `ResultSet` getMethods are used to obtain the data. In this case, we use `getString(1)` to obtain the value of the first column as a string. After obtaining the information, the result set, statement, and connection objects are closed. When using JDBC, it is important to close these objects to free database resources that were used by those objects.

Example 5-39 JDBC example using Java

```
import java.sql.*;

public class Sqlquery {
    public static void main (String[] args) {
        try {
            Connection connection = DriverManager.getConnection("jdbc:db2:*local");

            Statement statement = connection.createStatement();

            ResultSet rs = statement.executeQuery("SELECT CURRENT SERVER FROM
SYSIBM.SYSDUMMY1");
            rs.next();
            String currentServer = rs.getString(1);
            rs.close();
            statement.close();
            connection.close();

            System.out.println("Current system is "+ currentServer);
        }
    }
}
```

```
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

The program is compiled by using `javac` and then run by using Java from QSH. The output is shown in Example 5-40.

Example 5-40 Java JDBC output

```
$ java Sqlquery  
Current system is MYSYSTEM
```

Accessing DB2 for i from client applications

DB2 for i can be accessed from other platforms by using industry standard interfaces. For Java and Android environments, the IBM toolbox for IBM i and JTOpenLite JDBC driver are used. From Microsoft Windows environments, the i Access ODBC driver, i Access OLE DB provider, and i Access.NET provider provide access to DB2 for i.

Java and JDBC

Not only can JDBC be used on IBM i, but two JDBC drivers to access DB2 for i are available in the IBM Toolbox for Java, also released as the JTOpen open source project. For more information about the IBM Toolbox for Java JDBC driver, see the IBM Toolbox for Java manual found at <http://pic.dhe.ibm.com/infocenter/iseriess/v7r1m0/topic/rzahh/page1.htm>. For Android applications, the new JTOpenLite JDBC driver is also available. The JTOpenLite package was created to provide lightweight access to IBM i data from mobile devices that are running Java. For more information, see “JTOpenLite” on page 133.

To use the IBM Toolbox for Java JDBC driver with the previous Java program, the only line that must be changed is the `DriverManager.getConnection` statement. Because the JDBC program is not running on the IBM i, a user ID and password must be specified. Because we want to use the system name, user profile, and password that are passed to the Java program in our example, the new line becomes what is shown in Example 5-41.

Example 5-41 JDBC connections for remote access

```
Connection connection = DriverManager.getConnection("jdbc:as400://" + args[0],  
args[1], args[2]);
```

When it is running, the `jt400.jar` file must be included on the class path. Example 5-42 shows an example of running this program from AIX.

Example 5-42 Output when running the example program from AIX

```
> java -cp jt400.jar:. SqlqueryToolbox z1014p14 eberhard PASSWORD  
Current system is MYSYSTEM
```

Windows Platform APIs

IBM i Access provides interfaces for the Windows platform for access DB2 for i data. The interface that is used largely depends on the language that is used to develop the client application. The most generic interface, ODBC, is typically used by languages such as C or C++. The OLE DB interface is based on Microsoft Component Object Model (COM) and requires a language that is integrated with COM, such as Visual Basic. The .NET provider is used with .NET languages, the most prevalent being C#.

ODBC

ODBC is a common database interface that uses SQL as its database access language. Like the Call Level Interface, it provides a rich set of functions to connect to a database management system, run SQL statements, and to retrieve data. Also included are functions to interrogate the SQL catalog of the database and the capabilities of the driver. Unlike the OLE DB and .NET providers, ODBC can easily be used with any programming language on the Windows platform.

For more information about ODBC, see the “IBM i Access ODBC” section of the *IBM i Access for Windows: Programming* manual, found at:

<http://pic.dhe.ibm.com/infocenter/iseriess/v7r1m0/topic/rzaik/rzaikappodbc.htm>.

Example 5-43 shows an example that uses ODBC to access an IBM i database. This example is similar to the Call Level Interface example, with the following primary differences:

1. Different include files must be used for ODBC.
2. ODBC must use the `SQLSetEnvAttr` to specify the level of ODBC that must be used.
3. When connecting, the system name, user ID, and password must be included.

Example 5-43 ODBC access example

```
#include <stdio.h>

#include <sql.h>
#include <sqlext.h>

int main(int argc, char **argv)
{
    SQLHENV    henv = NULL;
    SQLHDBC    hdbc = NULL;
    SQLHSTMT   hstmt = NULL;
    char currentServer[20];
    SQLLEN currentServerLength;

    if(argc < 4)
    {
        fprintf(stderr, "%s [dsn] [userid] [pwd]\n", argv[0]);
        return -1;
    }

    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
    SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, 0);

    SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
    SQLConnect(hdbc, (SQLCHAR *) argv[1], SQL_NTS, (SQLCHAR *) argv[2], SQL_NTS,
              (SQLCHAR *) argv[3], SQL_NTS);

    SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
    SQLExecDirect(hstmt, (SQLCHAR *) "SELECT CURRENT SERVER FROM
SYSIBM.SYSDUMMY1", SQL_NTS);
    SQLBindCol(hstmt, 1, SQL_C_CHAR, currentServer, sizeof(currentServer)-1,
              &currentServerLength);
    SQLFetch(hstmt);

    printf("Current user is %s.\n", currentServer);
}
```

```

SQLFreeStmt(hstmt, SQL_CLOSE);
SQLFreeHandle(SQL_HANDLE_STMT, hstmt);

SQLDisconnect(hdbc);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, henv);

return 0;
}

```

Object Linking and Embedding Database

Object Linking and Embedding Database (OLE DB) is a set of interfaces that expose data from various sources. OLE DB interfaces provide applications with uniform access to data that is stored in diverse information sources, or data stores. These interfaces support the amount of DBMS functionality that is appropriate to the data store, enabling the data store to share its data. The IBM i Access for Windows OLE DB Provider component gives IBM i programmers record-level access interfaces to IBM i logical and physical DB2 for i database files. In addition, they provide support for SQL, data queues, programs, and commands.

For more information about OLE DB, see the “OLE DB Provider” section of the *IBM i Access for Windows: Programming* manual, found at:

<http://pic.dhe.ibm.com/infocenter/iseriess/v7r1m0/topic/rzaik/rzaikoledbprovider.htm>

Example 5-44 shows an example, which is written in Visual Basic, of how the OLE DB can be used to run the query that is used by the previous example. In this program, an ADODB connection object is created. The connection is then used to access a database by using the open method. In the open method, the OLE DB provider, IBMDASQL, is specified along with the information that is needed by the provider to connect to the database. In this case, the name of the system, along with the user ID and password, is supplied. When the connection is established, the **Execute** method is used to run the query and obtains a record list. The record set is then examined and the fields are then displayed.

Example 5-44 OLE DB example written in Visual Basic

```

Module Module1
    Public connection As Object
    Public recordset As Object

    Sub Main()
        connection = CreateObject("ADODB.Connection")
        connection.Open("Provider=IBMDASQL;Data
Source=MYAS400;User=MYUSER;Password=MYPASS")

        recordset = connection.Execute("SELECT CURRENT SERVER FROM
SYSIBM.SYSDUMMY1")
        Dim fd As Object
        For Each fd In recordset.Fields
            Console.WriteLine(fd.Value)
        Next fd

        recordset.Close()
        connection.Close()
    End Sub
End Module

```


End Sub
End Module

.NET

IBM i Access for Windows .NET Provider allows .NET managed programs, typically written in C#, to access the IBM i database files by using SQL. This provider consists of a set of classes and data types that provide access to Connection, Command, DataAdapter, and DataReader functions as defined and supported by the ADO.NET architectural model. Using ADO.NET, applications can connect to DB2 for i and retrieve, manipulate, and update data. For more information about .NET, see the “Windows .NET Provider” section in the *IBM i Access for Windows: Programming* manual, found at:

<http://pic.dhe.ibm.com/infocenter/iseriess/v7r1m0/topic/rzaik/rzaikdotnetprovider.htm>

Example 5-45 is an example, which is written in C#, that uses .NET to run a simple query. In this example, the application must explicitly create an `iDB2Connection` object. It then specifies the database name, user ID, and password using the `ConnectionString` property. The connection is then established using the `Open()` method. When the method is opened, a command is created. The query is then specified by using the `CommandType` and `CommandText` properties. The query is run by starting the `ExecuteReader` method of the command, which returns a reader object from which the results can be obtained.

Example 5-45 .NET access DB2 written in C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using IBM.Data.DB2.iSeries;
using System.Data;

namespace DotNetConsole
{
    class Program
    {
        static void Main(string[] args)
        {
            try {
                iDB2Connection iDB2MainConn = new iDB2Connection() ;
                iDB2MainConn.ConnectionString = ("DataSource="+args[0]+
                                                ";UserID=" + args[1] +
                                                ";Password=" + args[2]);

                iDB2MainConn.Open();

                iDB2Command cmd = iDB2MainConn.CreateCommand();
                cmd.CommandType = CommandType.Text;
                cmd.CommandText = "select current server from sysibm.sysdummy1";
                IDataReader reader = cmd.ExecuteReader();
                while (reader.Read())
                {
                    Console.WriteLine(reader.GetString(0));
                }
                reader.Close();
                cmd.Dispose();
                iDB2MainConn.Close();
            }
        }
    }
}
```

```

        } catch (Exception e ) {
            Console.WriteLine(e.StackTrace);
        }
    }
}
}

```

Example 5-46 shows an example of running the program from the command line.

Example 5-46 .NET example output

```

C:>DotNetConsole SYSTEM EBERHARD PASSWORD
MYSYSTEM

```

DRDA solutions

All IBM relational database products, including DB2 for i, include IBM Distributed Relational Database Architecture™ (DRDA) support for distributed relational database processing. DRDA defines protocols for communication between an application program and a remote relational database. From an IBM i perspective, this means that DB2 for i can be accessed by any client that uses DRDA. Typically, an application uses some type of middleware that converts application requests into DRDA requests that are sent to the server.

On IBM i, DRDA is used to connect to the remote database by using the Relational Database Directory. Using the **WRKRDBDE** (Work with Relational Database Directory Entries) command, it is possible to add connection to remote servers. When a Relation Database Directory Entry is created, any of the previously mentioned server-side database mechanisms can be used to access the remote systems. For example, the native JDBC driver can access a remote system by using the `jdbc:db2:REMOTESYS` URL, where *REMOTESYS* is the name of the remote system configured using **WRKRDBDIRE**. More information about using configuring relational database entries can be found in the *Distributed Database Programming Manual*, found at:

<http://pic.dhe.ibm.com/infocenter/iseriess/v7r1m0/topic/ddp/rballkickoff.htm>

DRDA is also being used by Oracle through the Oracle Database Gateway for DRDA product. Using this gateway, DRDA servers are accessible from any Oracle supported platform and network environment. For more information, go to the following website:

<http://www.oracle.com/technetwork/database/enterprise-edition/tg4drda-097332.html>

DB2 Connect

IBM DB2 Connect™ is an IBM product that provides connectivity to mainframe and midrange databases from the Linux, UNIX, and Windows operating systems. It provides Call Level Interface, ODBC, and JDBC drivers that can be used on various platforms. More information about IBM DB2 Connect can be found at the following website:

<http://www.ibm.com/software/products/en/db2-connect-family>

Summary

The information that is stored in DB2 for i is readily available by using various programming interfaces. On IBM i itself, all ILE language can use embedded SQL and Call Level Interface. Java applications can use the native JDBC driver and PHP applications can use the built-in `ibm_db2` driver. From other platforms, Java applications can use the IBM Toolbox for Java JDBC driver to connect to DB2 for i. On the Windows platform, DB2 for i can be accessed through ODBC, OLE DB, or .NET. Finally, DB2 for i supports DRDA, which allows access by using various middleware components.



User interface

This chapter is an overview of user interface (UI) modernization techniques. Given that it is the normal means for interaction with IBM i applications, UI is the first and most obvious point of pain when using traditional applications. Requests to change the user interface typically result in this being the first step in many IBM i modernization paths.

When UI modernization completes, and if user complaints subside, IT departments often believe that their modernization is complete. This is a modernization myth. UI modernization is one small part of an IBM i modernization strategy. Given this fact, this chapter does not cover such important topics as security, infrastructure, database, or other modernization steps.

6.1 Text-based user interface

The traditional UI for any IBM i application is a green screen emulator program running on a client device. This interface is often referred to as a Character User Interface (CUI) or Text-based User Interface (TUI).

This interface is limited to delivering one screen of information at a time to a client, either 24 rows and 80 columns or 27 rows and 132 columns. This data is transferred between the host and the client by using a Telnet connection through the TN5250 protocol. All conversations are stateful and in block mode. An entire screen of data is sent to the client, and the host server waits for a response. When a function key is pressed, the entire screen of data, including any user data entry, is sent back to the host, which can then continue program execution.

Here are the advantages of using an emulator for user interface access to an IBM i application:

- ▶ Speed
 - Response time
 - Data entry
- ▶ Type-ahead keyboard buffering
- ▶ Familiar interface

Here are the disadvantages of using an emulator for user interface access to an IBM i application:

- ▶ Not designed for a mouse.
- ▶ No graphical user controls.
- ▶ Limited amount of information is displayed.
- ▶ An emulator client must be installed on the user device.
- ▶ It is difficult to teach to new users.
- ▶ Limited data types.

Figure 6-1 on page 175 shows an example of a typical 24 x 80 green screen application at the client waiting for user input.

```

MAIN                                IBM i Main Menu                                System:  C104730F
Select one of the following:

  1. User tasks
  2. Office tasks
  3. General system tasks
  4. Files, libraries, and folders
  5. Programming
  6. Communications
  7. Define or change the system
  8. Problem handling
  9. Display a menu
 10. Information Assistant options
 11. IBM i Access tasks

 90. Sign off

Selection or command
====> _

F3=Exit  F4=Prompt  F9=Retrieve  F12=Cancel  F13=Information Assistant
F23=Set initial menu

```

Figure 6-1 Typical 24 x 80 green screen interface

The architecture of the IBM i 5250 interface includes technology to transform the green screen application to a 5250 telnet data stream. This technology is part of IBM i, and includes the following components:

- ▶ User device (TN5250 emulator client)
- ▶ IBM i host
 - Application program
 - Display file
 - Handler (Display manager (included with IBM i))
- ▶ Telnet server

Figure 6-2 shows the architecture of an IBM i TN5250 interface.

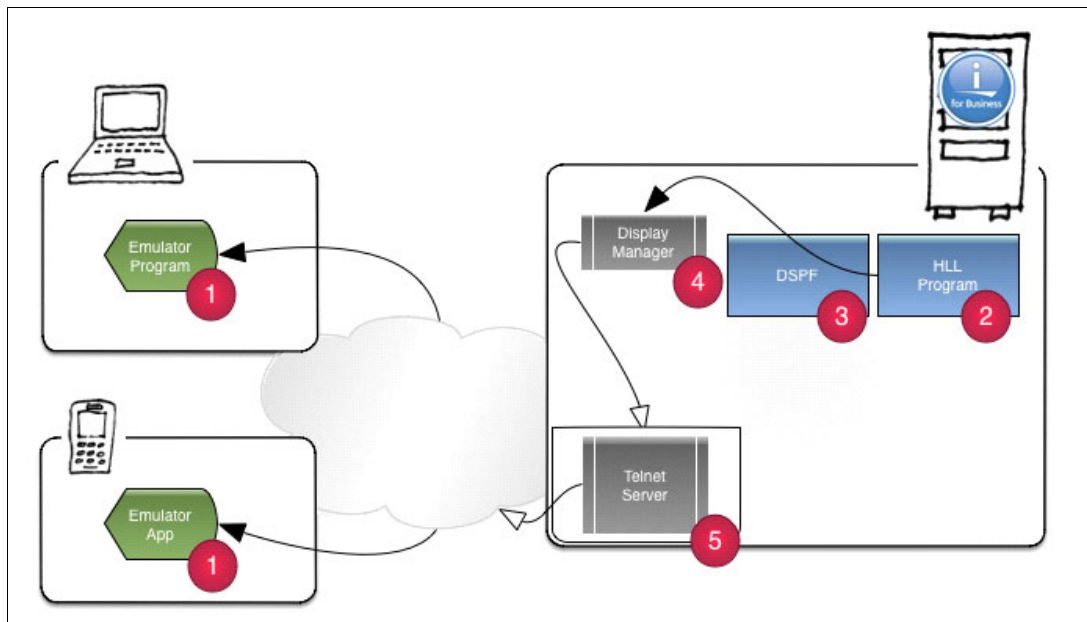


Figure 6-2 Representation of IBM i TN5250 architecture

There are many approaches to modernizing an IBM i user interface. This chapter outlines the four most common techniques:

- ▶ Browser
- ▶ Rich client
- ▶ Custom application
- ▶ Service-enabled application

6.2 Browser user interface

Delivering the 5250 interface to a browser provides access to IBM i applications without installing a specific client to the user device. Given that modern user device operating systems, including Windows for desktop and mobile, desktop Linux, OS X, iOS, Android, Chromium, and so on support and include a browser, any device that has access to the HTTP server can access IBM i applications.

This technique requires an HTTP server to deliver the HTML data stream to the user device. For the shipped IBM i solution, the HTTP server is run on IBM i. Third-party vendor applications can use an HTTP server on IBM i, or use an HTTP server on another platform.

To transform the green screen application user interface to HTML, an application is required. The typical solution is to run this transformation application on an application server behind the HTTP server.

The shipped IBM i web browser interface is IBM i Access for Web. For more information about this interface, go to the following website:

<http://www.ibm.com/systems/power/software/i/access/web.html>

In addition to providing an HTML interface to IBM i applications, IBM i Access for Web also runs batch commands and provides access to databases, integrated file systems, printers, printer output, and messages. Figure 6-3 on page 177 shows an example of the configuration page for IBM i Access for Web.

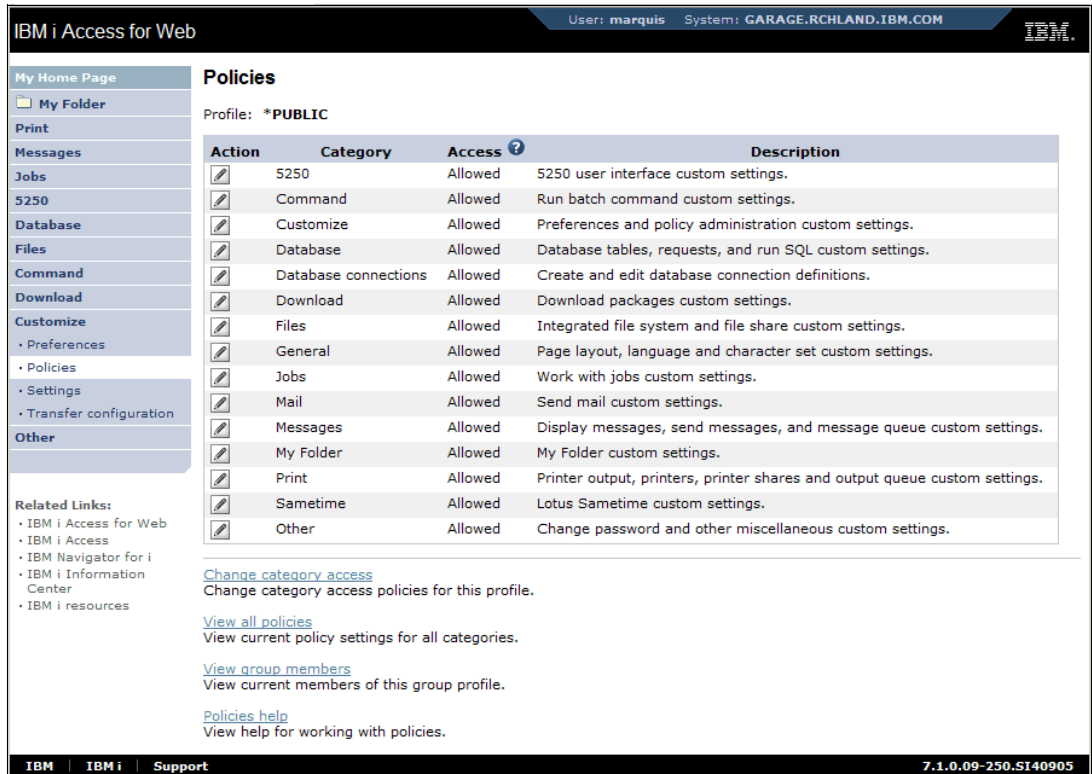


Figure 6-3 An example of the configuration page for IBM i Access for Web

Third-party vendor applications that provide web browser interfaces often provide additional IBM i functions with their tools. Normally, these functions are access to messages, access to spooled files for conversion to PDF, email, and so on.

When accessing the transformed IBM i 5250 application, the communication from the HTTP server to the user device browser is stateless. The stateful communication to the running green screen application is normally maintained by the application server.

For this technology, it is common to access the HLL programs by using the web-facing APIs that are provided in the IBM i operating system.

The architecture of the shipped IBM i web browser interface has the following additional components beyond the TN5250 architecture:

- ▶ User device (browser)
- ▶ IBM i host
 - Application program
 - Display file
 - Handler (web-facing APIs (included with IBM i))
 - Transformation application
 - Application
 - Application server
 - HTTP server

Figure 6-4 shows a representation of a shipped browser user interface architecture.

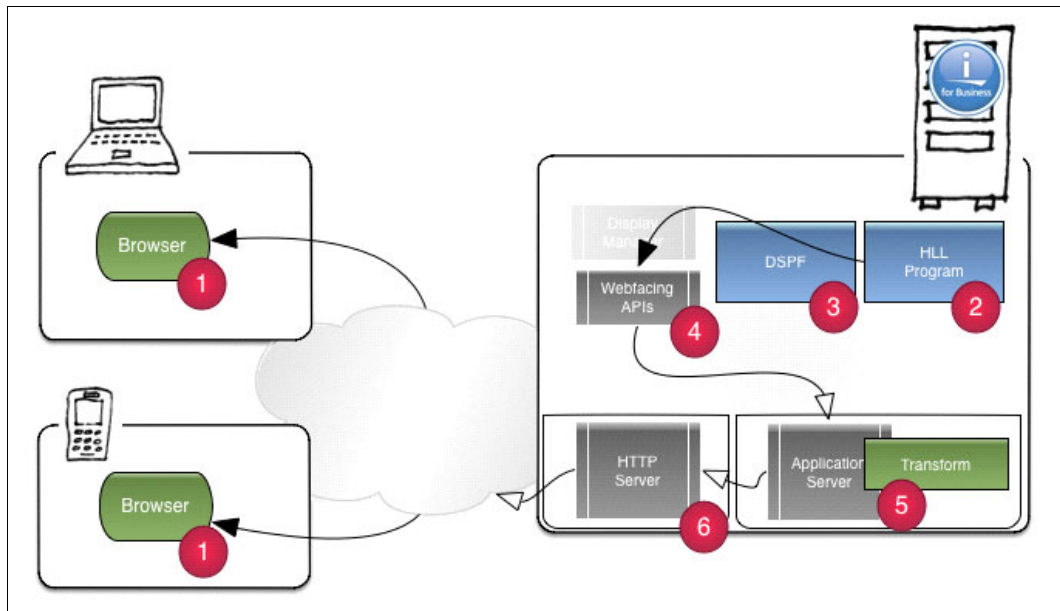


Figure 6-4 A representation of a shipped browser user interface architecture

Figure 6-5 is an example of a third-party browser solution that is delivered to a tablet.



Figure 6-5 Third-party browser solution that is delivered to a tablet

The architectures of some third-party IBM i web browser interfaces include many components. Here is an example of such an architecture:

- ▶ User device (browser)
- ▶ IBM i host
 - Application program
 - Display file
 - Handler
 - Display manager (included with IBM i)
 - Web-facing APIs (included with IBM i)
- ▶ Alternative server
 - Transformation application
 - Application
 - Application server
 - HTTP server

Figure 6-6 is an example of a browser user interface architecture.

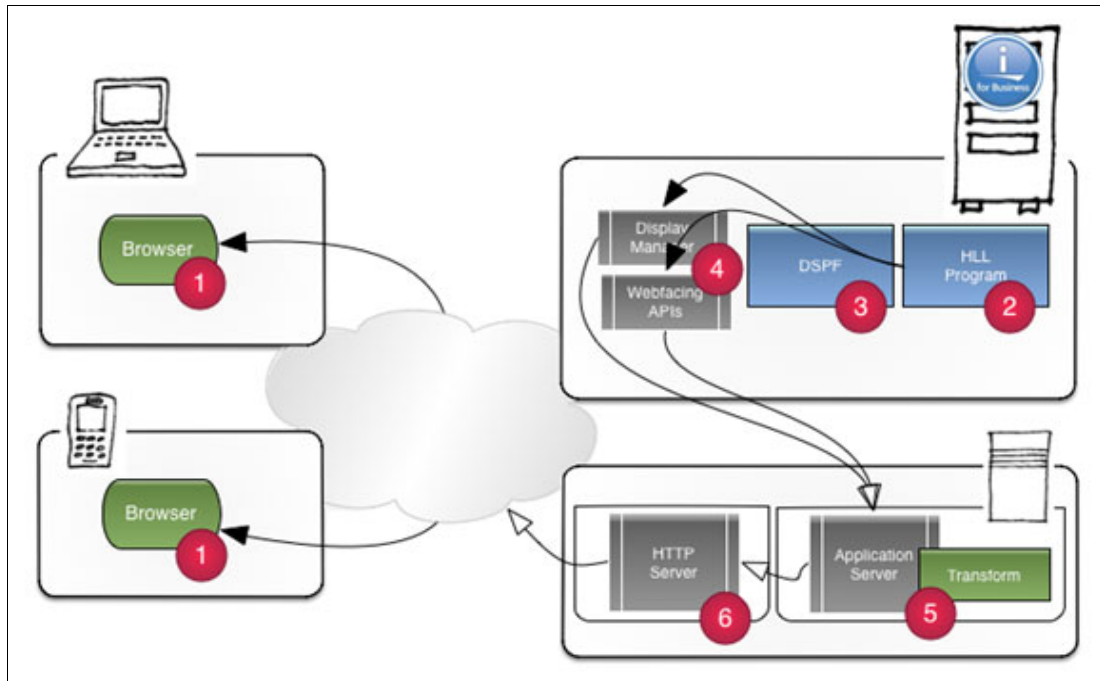


Figure 6-6 Representation of an example browser user interface architecture

To extend the user interface further, IBM has provided, within IBM i, a technology named RPG Open Access. For all RPG file operations, a custom handler can be written to enhance the communication, in any HLL, by any IBM i developer. For Display Files, a handler can be written to be used in place of the Display Manager and the web-facing APIs. The transformation application can connect to the original Display Manager, the web-facing APIs, or a specific Open Access handler program that is written to replace the Display File interface.

The architecture of some third-party IBM i web browser interfaces with RPG Open Access includes many components. Here is an example of such an architecture:

- ▶ User device (browser)
- ▶ IBM i host
 - Application program
 - Display file
 - Handler (RPG Open Access Handler)

- ▶ IBM i host or alternative server
 - Transformation application
 - Application
 - Application server
 - HTTP server

Figure 6-7 is a representation of an example RPG Open Access handler architecture.

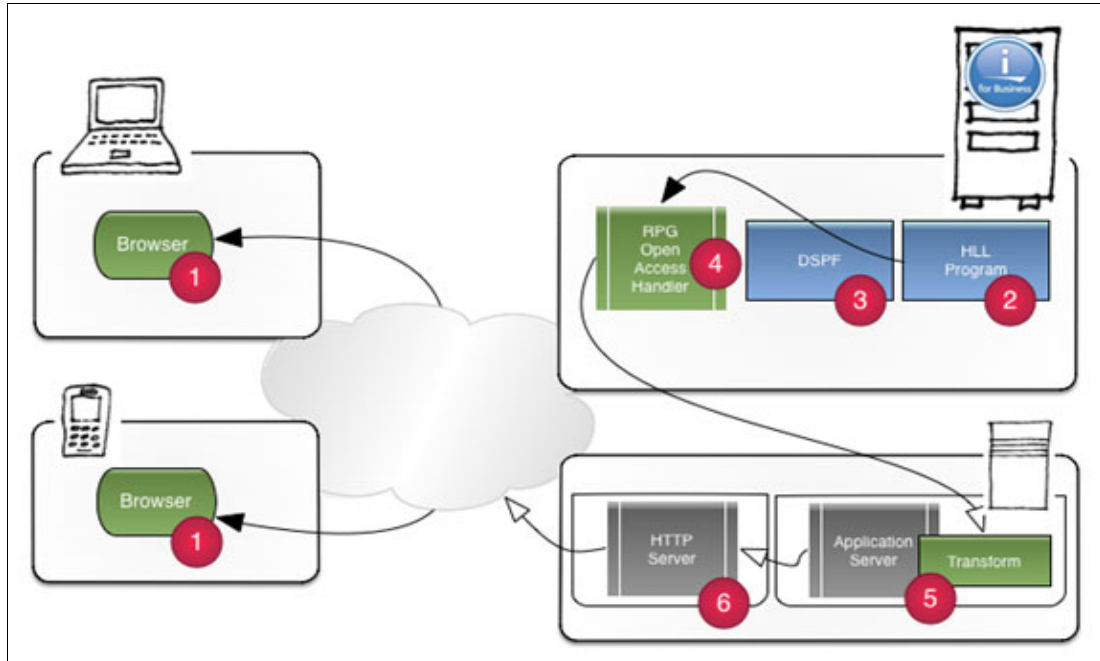


Figure 6-7 Representation of an example RPG Open Access handler architecture

Here are the advantages of using a browser for user interface access to an IBM i application:

- ▶ No client installation is required.
- ▶ Link function to access data and information.
- ▶ GUI access to other IBM i functions.
 - Messages
 - Spooled files
 - Database

Here are the disadvantages of using a browser for user interface access to an IBM i application:

- ▶ No type-ahead keyboard buffering
- ▶ Limited graphical user controls
- ▶ Limited amount of information displayed
- ▶ Inability to tightly integrate other applications

6.3 Rich-client user interface

Beyond the basic browser GUI delivery, the next choice for modernizing IBM i applications is a rich client. This client runs natively on the user device, and can communicate with the IBM i host in several ways. The TN5250 telnet communication can be used, the HTTP and application server can deliver content, and RPG Open Access can be used to connect to the green screen application programs.

The first third-party UI modernization tools used screen-scraping technologies and focused on delivering a GUI to a Windows desktop. In the decades since, screen-scraping technology has evolved dramatically. Some third-party vendor tools still provide a simple transformation of the 5250 data stream, but most rich-client tools transform the data stream to a modern object-oriented, event-driven application.

Most rich-client solutions are specific to one type of user device. Coding native applications for Android and Windows, for example, requires a different set of tools, a different skill set, and a different deployment mechanism. Some toolsets are available to build native applications in one language and create a deployable version for multiple platforms, but typically these solutions are available for only the functions that are common to all platforms.

If a solution is specific to a user device platform, it can take advantage of advanced functions on that platform. For example, a Windows rich client should be able to take advantage of the .NET framework, and be able to integrate to other running desktop applications, such as email, word processing, and spreadsheets. For another example, a smartphone rich-client application should be able to take advantage of the phone device tools and applications, such as GPS, mapping, gyrometer, contacts, and the camera.

Rich-client solutions can also be embedded in other custom applications, and integration to those other applications is a key benefit. The integration depends on the modernization and development tools that are selected and the user devices that access the modern application.

Figure 6-8 shows an example of a rich-client solution that is running in Windows and embedded in Microsoft Outlook.

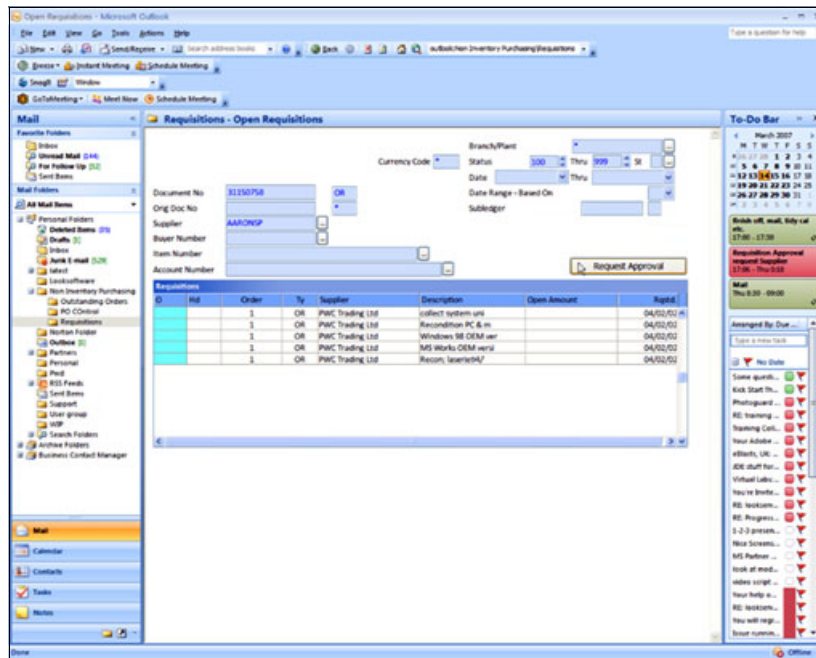


Figure 6-8 Example of a rich-client solution running in Windows and embedded in Outlook

The architecture of some third-party IBM i rich-client interfaces includes many components. Here is an example of such an architecture:

- ▶ User device (rich client)
- ▶ IBM i host
 - Application program
 - Display file

- Handler
 - RPG Open Access Handler
 - Display manager (included with IBM i)
 - Web-facing APIs (included with IBM i)
- ▶ IBM i host or alternative server
 - Telnet server
 - Transformation application
 - Application
 - Application server
 - HTTP server

Figure 6-9 shows a representation of a rich-client solution architecture

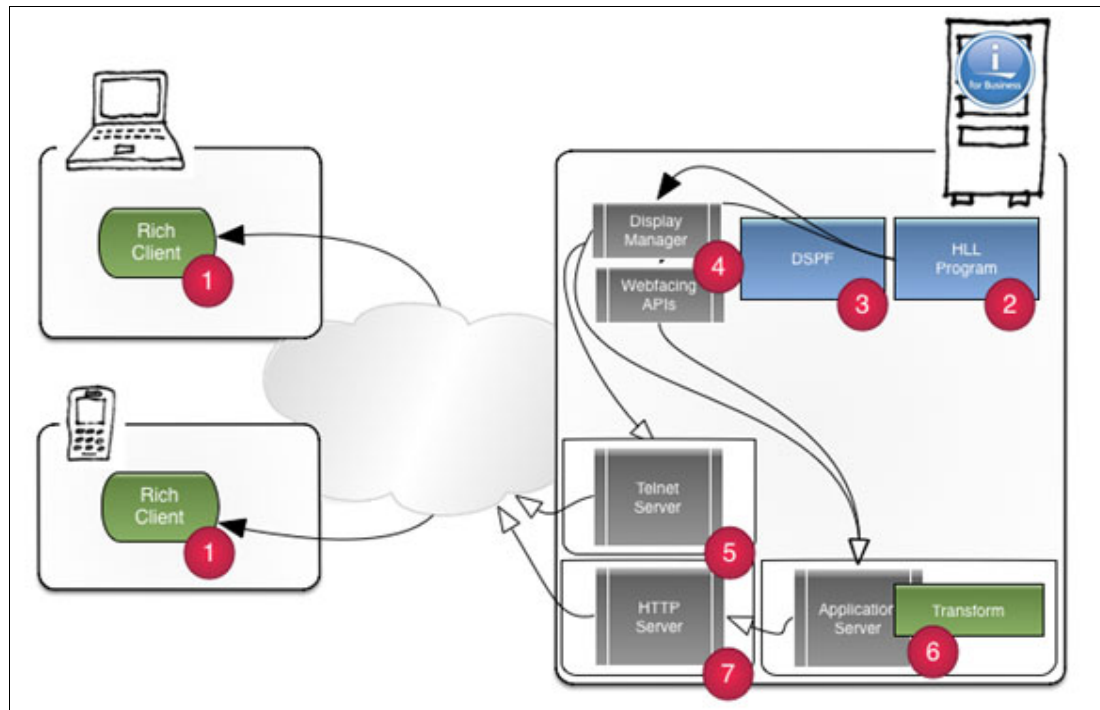


Figure 6-9 Representation of a rich-client solution architecture

Here are the advantages of a rich client for user interface access to an IBM i application:

- ▶ Type-ahead keyboard buffering (some solutions)
- ▶ Link function to access data and information
- ▶ Tight integration to other native and IBM i applications
- ▶ Rich GUI controls

Here are the disadvantages of using a rich client for user interface access to an IBM i application:

- ▶ Client installation is required.
- ▶ Requires a custom client for each type of user device.
- ▶ Custom deployment for each device type is required.

6.4 Custom application user interface

Some companies choose to build modern applications that run directly on a user device to provide the user interface to IBM i applications. This task requires developer skills that are based on the types of user devices that access the modern application.

Custom client applications are normally built to provide access to IBM i green screen applications while delivering a modern user interface, providing additional functions and integrating data from multiple locations.

Figure 6-10 shows an example of a third-party custom application running on Windows.

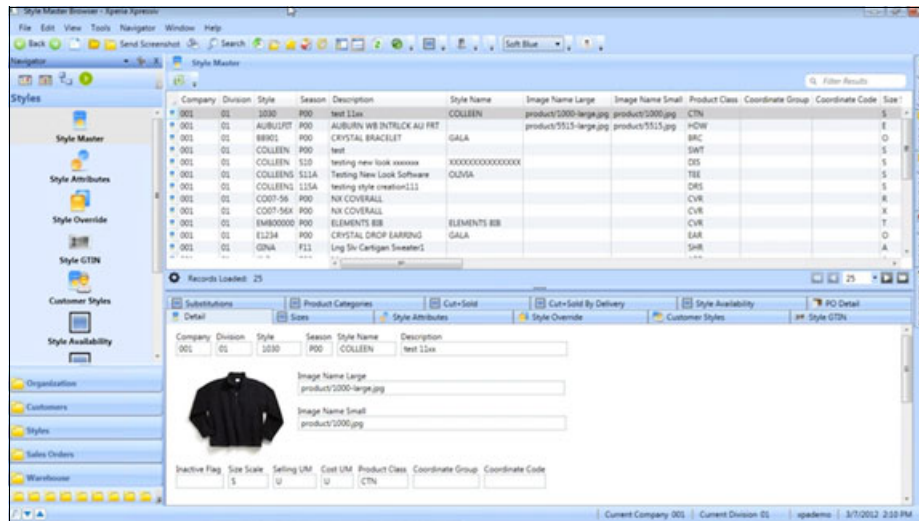


Figure 6-10 Example of a third-party custom application running on Windows

To access the IBM i application, there are several means of communication. Primarily, IBM i servers are used for that communication. Here are some of those servers:

- Telnet server** Provides a connection to a client that can interpret a TN5250 data stream.
- Web application server** Configured behind an HTTP server to deliver an application to a browser client.
- FTP server** Allows an FTP client to connect to the IBM i host and upload or download data.
- RPC server** Provides a connection to run IBM i HLL programs, with the ability to send and receive parameters.
- Database server** Provides a connection to the IBM i DB2 database, and is fully SQL-capable.
- Java toolbox** Provides Java native methods and access methodology for Java applications to access IBM i native data.

The architectures of some third-party IBM i custom application interfaces include many components. Here is an example of such an architecture:

- ▶ User device (custom application)
- ▶ IBM i host
 - Application program
 - Display file

- DB2 database file
- Handler
 - RPG Open Access Handler
 - Display manager (included with IBM i)
 - Web-facing APIs (included with IBM i)
- ▶ IBM i host
 - Telnet server
 - Application server
 - Transformation application
 - Web services delivery application
 - HTTP server
 - Other servers
 - FTP server
 - RPC server
 - Database server

Figure 6-11 shows a representation of an example custom client solution architecture.

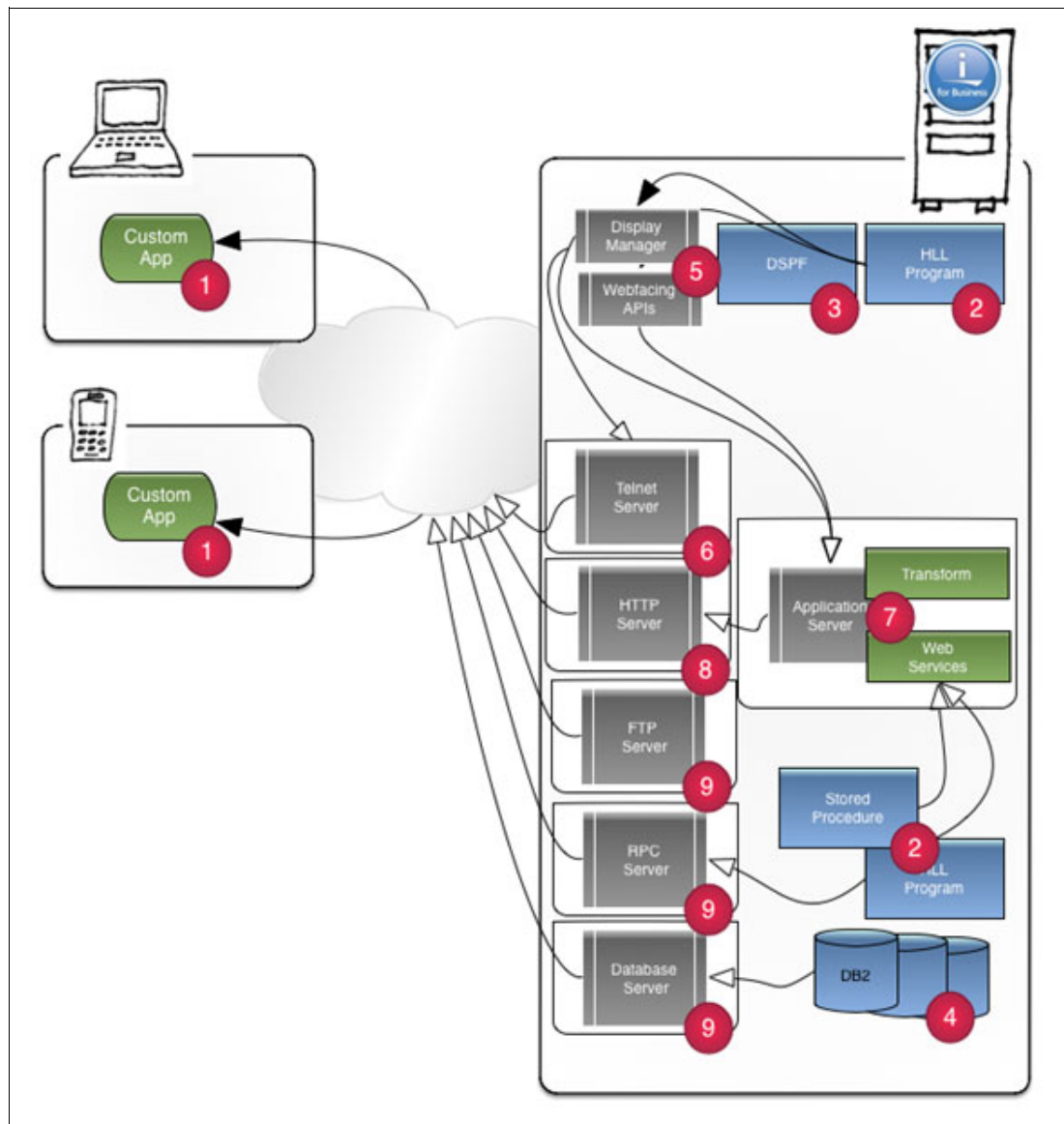


Figure 6-11 Representation of an example custom client solution architecture

Many custom client applications can use web services, connecting to other databases and connecting to other servers to provide a rich integration capability of applications within and from outside the company infrastructure.

To support the consumption of web services, custom web services are built and exposed from the IBM i host. These functions are available with IBM i, and include the ability to configure IBM i stored procedures and service programs and web services by using a wizard. Third-party vendors also provide tools to expose web services from IBM i applications, and the approaches are diverse.

The architectures of some IBM i web service interfaces include many components. Here is an example of such an architecture:

- ▶ Other applications (web service consumer)
- ▶ IBM i host
 - Application program
 - Application server (web services delivery application)
 - HTTP server
- ▶ IBM i host or alternative server
 - Application server (web services delivery application)
 - HTTP server

Figure 6-12 shows a representation of an example web services architecture.

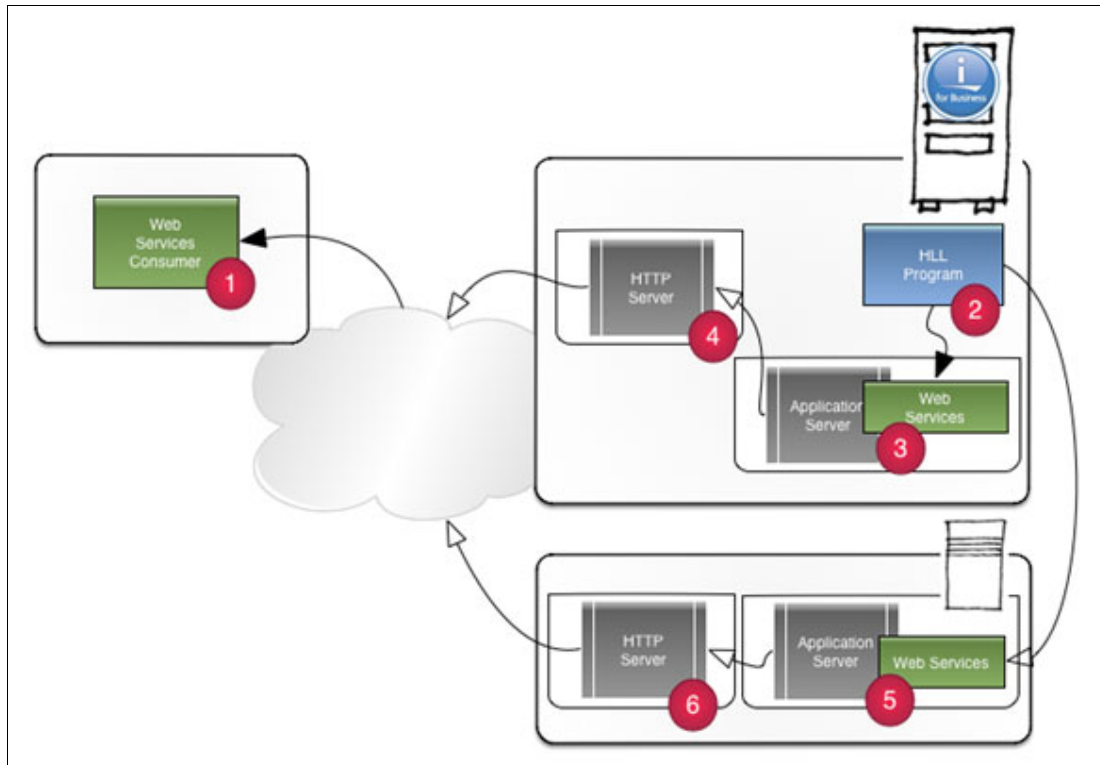


Figure 6-12 Representation of an example web services architecture

Here are the advantages of using a custom client application for user interface access to an IBM i application:

- ▶ Very rich functionality
- ▶ Tight integration to other native and IBM i applications
- ▶ Integration to other servers and databases
- ▶ Rich GUI controls

Here are the disadvantages of using a browser for user interface access an IBM i application:

- ▶ Client installation is required.
- ▶ Requires a custom client for each type of user device.
- ▶ Custom deployment for each device type is required.

6.5 Summary

No single user interface modernization solution is correct for every company, or every IBM i application suite. Although UI modernization is often the first step for many companies, consideration must be made to include UI modernization as just one part of a complete IT strategy or modernization roadmap.

Other tools and technologies that are chosen for modernization might dictate or guide your choice in a UI modernization approach. Several choices might be required for different user devices, for different IBM i application suites, and different timelines. A single vendor rarely provides an entire modernization suite, or an entire solution set for UI modernization. Custom development might be required, and it is important to ensure the skill set of your developers is kept up to date.

With more modern applications delivering all or part of their solutions to a browser, and as browser clients become more sophisticated and function-rich, it is important to learn web development languages such as HTML, CSS, and JavaScript. Learning and understanding modern development techniques is important, especially visual development, object-oriented, and event-driven development tools. Adopting frameworks for consistency and reduced maintenance must be given high consideration.

Designing the user interface is also important. Typically, green screen developers have little experience with graphical design, and often do not understand how best to design and build an effective user interface. Engaging skilled graphical designers can result in elegant and efficient modern applications. It is important for IBM i developers to have a minimum understanding of graphical design, at the least to engage and communicate with the graphical design resources. More often than not, a user interface that is designed by a programmer is not simple to use or easy to understand, and requires additional effort for users when performing their work.

User experience is also a key concept that must be understood by a modernization team when building a modern user interface. Understanding how users perform their tasks, how users interact with each type of user device, how to reduce the number of steps that is required, and how to deliver the correct information at all times improves the user experience for the modern application. Collecting and reviewing analytics of the application use can also contribute to an improved application design and enhance the user experience.



Modern RPG

Is “modern RPG” an oxymoron? Certainly not. There is a modern form of RPG and modern styles of using the latest iteration of the language.

In *Who Knew You Could Do That with RPG IV? A Sorcerer's Guide to System Access and More*, SG24-5402, you can find an early history of the RPG IV language. However, that book was written in 2000, describing the then current version of RPG at Version 4 Release 4. Since that time, the world has changed dramatically and RPG has changed with it.

The most obvious change is the fact that today's RPG can be written in free format. RPG was a columnar language for most of its long history. That was even true at the time of the earlier publication in Version 4 Release 4. Not long after the publication of that book came Version 5 Release 1, and with it, the ability to write RPG logic in free format. That not only represented a huge change in the usability and readability of RPG, it also enabled better usage of the source in the language to support other advances, such as complex expressions and qualified names. More recently in IBM i 7.1, TR7 RPG is free format, even in declarations of data and files.

As exciting as a free format RPG language is, modern RPG entails much more than writing code in a readable manner. This chapter explores many aspects of modern RPG.

7.1 The definition of modern RPG

Today's RPG is an exceedingly powerful and efficient language, which can support the most modern of programming styles and modern user and application interfaces. Because of RPG's long history, there is a wealth of RPG applications and RPG shops in the world today. Much of the existing RPG code exists in earlier forms of the language, that is, those forms before RPG IV. Even code that is written in RPG IV, however, might still not be modern in style, structure, or data access.

To some extent, “modern” is in the eye of the beholder. There is no definitive standard of what makes an RPG application modern. Among any group of RPG users, there is almost certainly disagreement about the precise details of the definition of modern RPG. As the language, database, and system changes, such a definition needs to change with it.

However, there are at least some aspects of modern RPG that most RPG developers, and those of us writing this book, tend to agree form the foundation of modern RPG applications. Here is a list of the most commonly agreed upon things that constitute modern use of RPG:

- ▶ Only the RPG IV language is modern, which means that the source member type must be RPGLE instead of RPG and not RPG36 or RPG38. Although there might not be consensus on all the points in our definition, the usage of the RPG IV language is typically considered the litmus test of a modern RPG application.
- ▶ Modern RPG code is written in free format. Although the ability to write the entire program that way is relatively new (as of this writing), the ability to write logic in free format has been available for many years. So, a modern RPG application should have at least its logic written in free-form.
- ▶ Modern RPG applications use a modular design to enhance the reusability and flexibility of the code. Modularity is enhanced by the usage of the Integrated Language Environment (ILE), especially its ability to store reusable pieces of code in service programs. RPG subprocedures are used to create the callable functions that go into the service programs and are also used to replace subroutines for internal code structure within a module.
- ▶ Modern RPG code follows a set of preferred practices for both application design and coding. We describe the details of some of those preferred practices later in this chapter.

Of course, any RPG program or application falls somewhere in a spectrum between old-fashioned and modern. We do not intend for anyone to use our definition of modern RPG or our list of preferred practices as a strict standard for categorizing any application as modern versus old-fashioned. For example, you might consider an application to be modern even if it contains a few subroutines and it fits most of the other criteria. Our definition and preferred practices are meant to be used as guidelines for RPG developers to consider as candidates for their own standards and practices as they move forward with their modernization efforts.

7.2 The business value of modern RPG

Today's RPG IV language is a highly efficient and flexible language that is designed for today's business applications. Flexibility of data access with DB2 for i is provided through the ability to use powerful industry-standard SQL statements or the unparalleled simplicity of built-in, record-level access for simpler data requirements. A built-in XML parser simplifies dealing with incoming XML data. RPG's excellent native support of decimal arithmetic allows for highly efficient handling of business-oriented mathematics. Its integration with system functions and with other languages, such as PHP and Ruby, allow developers to easily expand applications beyond what RPG's built-in support provides. With the help of RPG Open Access, many of these interfaces can be integrated into the language.

The fact that RPG code that was originally written decades ago, perhaps for a System/36 or System/38, still runs today on IBM i is often considered a huge advantage. Shops are not required to rewrite the code to take advantage of the latest in hardware technology because of the Technology Independent Machine Interface (TIMI) and the upward compatibility of the RPG compilers. It is often mentioned as a great feature for investment protection.

However, just because the code still runs on today's systems does not mean that it is providing the value to the business that it could or should. Indeed, code that is written in older, less capable versions of the RPG language are considerably harder to maintain and even harder to update to use new application technologies and interfaces. Even if the code is written in (or converted to) RPG IV from RPG III or earlier, if the style and techniques that are used are contrary to modern recommendations, the time that is needed to make even relatively simple modifications, additions, or adaptations to the code can be dramatically increased, increasing the cost of maintaining the application.

Not all parts of every application need much maintenance. So, TIMI is helpful in that it gives shops the flexibility to modernize applications (or parts of them) as needed to better serve the business needs and leaving other applications in their original state.

However, you must consider the old adage of "If it is not broken, do not fix it". If an application is taking longer than necessary to maintain or if it is difficult or impossible to add functions that can serve the business better, it is costing the company money. Each shop must decide for their own situation what constitutes "broken" and which applications must be modernized and to what extent.

Even if a shop decides not to modernize some of their existing code, there is no reason not to use modern RPG for all new code being written in that shop.

The modern RPG IV language can often do more with much less code. Consider, for example, the amount of logic that is necessary to determine the number of days between two dates. Using date data types and the built-in functions to operate on them, that operation can be done in a single statement and can even be part of a larger expression. Compare that setup to the dozens of statements it often took to do that simple calculation using older RPG techniques. There are countless other examples of using modern RPG language features to dramatically simplify logic and increase efficiency, both runtime efficiency, and more importantly, programmer efficiency.

Modern RPG enables more readable code because of its support for longer, more meaningful names, total free-format methodology, and its support for indenting to make logic flow more obvious. More readable code is inherently more understandable code. The easier it is to understand the code, the easier it is to maintain it. Therefore, maintenance can be done faster and with fewer mistakes introduced. So, more readable code often means more reliable code and more productive maintenance. Because shops spend much more time maintaining existing code than writing new code, the readability of the code can have a huge payback.

When looking for developers to fill positions in RPG shops, the usage of modern RPG increases the pool of viable candidates. Modern, free-form, and modular RPG has much in common with other languages, such as PHP or Java. Therefore, many RPG shops find that they can hire developers with skills in other languages (coupled with the usage of modern development tools such Rational Developer for i) and they can easily, and happily, learn the RPG language skills that are necessary to become productive quickly. There is no longer a need to limit your search to those developers who already have experience with RPG. Hiring developers with experience in other environments might also be a good way to bring in some new skills into the shop, such as HTML5, CSS, and JavaScript.

7.3 Moving to modern RPG

Because of the long history of the language, there are many examples of old-style RPG in almost every IBM i development shop. Moving from an old to a modern application design is something that cannot be completely automated, although there are tools that can help with the process of modularization and restructuring applications.

However, moving older style RPG code into RPG IV and to free format can be substantially, sometimes completely, automated. It is definitely not a productive use of a developer's time to manually convert logic when tools to automate the process are so readily available.

For programs that are still in RPG (or RPG36 or RPG38) source members, they must be moved to RPG IV. IBM supplies a rudimentary tool to convert the syntax of the code by using the Convert RPG Source (**CVTRPGSRC**) command. This command comes with the RPG compiler. Although the command supports converting only from RPG source member types, some shops have had some success converting RPG36 or RPG 38 members by changing the source member type before running the conversion. There almost certainly are some parts of the code that do not convert correctly. In those cases, you need manual intervention.

However, the **CVTRPGSRC** command makes almost no attempt to modernize the style of the code. Usually, it produces old-style RPG code that compiles by using the RPG IV compiler. The output still is in fixed-format RPG. Do not be misled that you can simply use **CVTRPGSRC** to create a modern RPG program.

There are some other tools that can help automate conversion to a more modern level of code, including moving to free format logic or even free format RPG. Different tools provide different degrees of automation and support different features.

One option to convert to free-form is built into the editor of the Rational Developer for i tool (and its predecessors). The developer can convert an entire source member by using the Convert All to Free-Form option from the Source menu. In addition, a section of logic can be highlighted and right-clicked, and you can select the Convert Selection to Free-Form option.

The conversion that is done by this built-in tool is rudimentary. For example, if it encounters an operation code that is not supported in RPG free format (such as **MOVE**, **MOVEL**, **ADD**, or **CALL**), it surrounds those statements with `/End-Free` and `/Free` directives, if you are using the new fully free-form implementation before TR7 of Version 7.1. So, it converts columnar RPG that is written in fairly modern style, but much of the logic of older RPG programs is not converted. The resulting mixture of free format and columnar RPG is often even less readable than the all columnar logic was. Fortunately, the free format conversion can be “undone” within the editor for those occasions. This tool can be helpful as a quick test to see how much manual effort might be necessary to move a specific source member to free-form.

However, for converting multiple source members and for doing a full- function conversion to free format, most RPG users find that using a separate tool is helpful. The following list provides a brief description of some of these tools:

► JCRCMDS

A free set of tools created by Craig Rutledge, which includes a set of utilities to convert fixed-format RPG logic to free format. For more information about this set of tools, see the following website:

<http://www.jcrcmds.com/>

JCRCMDS provides substantial assistance in converting older style RPG logic to free format syntax. For most older style RPG members, the conversion process involves multiple steps and requires some amount of manual effort. Using the tools in a documented sequence typically converts the bulk of the logic to free format.

There is a utility (JCR5FPRT) that you can use to highlight statements in a source member that do not convert to free format by using the companion JCR5FREE conversion tool. If there are many lines of code in that category, which is often the case, then the usage of the JCR4MAX utility converts to a better style of columnar logic, and the JCR4PROTO utility is used to generate prototypes and CALLP operations to replace **CALL** statements. After those steps, the JCR5FREE utility can be used to convert most of the code to free format. There are often some lines of code that still require manual change, but the bulk of the code is converted to use free format logic.

Although these utilities might require a few more steps than the other tools listed here, and do not integrate with the Rational Developer for i development tools, the fact that they are a no charge option makes them attractive to shops with budget constraints.

► Linoma Software RPG Toolbox

A chargeable set of tools for RPG developers, which includes the RPG Wizard, which can convert either RPG III (or PRG400) or RPG IV columnar code to free format, including the most recent free format H, F, D, and P specs. For more information about this tool, see the following website:

<http://www.linomasoftware.com/products/rpgtoolbox/rpgwizard>

The Linoma RPG Wizard offers a more streamlined approach to converting RPG code to a more modern form, including free format logic and declarations. It can start with code that is already in RPG IV (RPGLE member type) format or with code that is in RPG III (RPG member type) format. The tool can automatically convert most RPG programs in a single step. The **RPGWIZ** command converts operation codes that are unsupported in free format logic, including the creation of the prototypes that are necessary to replace **CALL** operations and inserting built-in functions when necessary to ensure similar behavior to the original code. Although there almost certainly are lines of code that require manual changes, the RPG Wizard can convert most, if not all, of the RPG through an extensive set of parameters of the **RPGWIZ** command.

The RPG Toolbox also now has an optional plug-in for Rational Developer for i, providing a much more robust solution than the free-form convert option that is already built into the Rational Developer for i editor. The plug-in includes the ability to convert to free-form and to indent free-form logic and uses preference settings to allow flexible control of conversion options. With the Rational Developer for i plug-in, selected blocks of code can be converted to free-form in addition to the ability to convert entire source members.

- ▶ **ARCAD Software ARCAD-Transformer RPG**

This is a chargeable tool that can plug into and integrate with Rational Developer for i. It can be installed as a stand-alone tool or as part of the ARCAD Pack for Rational. The ARCAD-Transformer RPG tool can be started from Rational Developer for i or from a native command line to automatically convert columnar RPGLE source code to free format.

For more information about this tool, see the following website:

<http://www.arcadsoftware.com/download2/softwaredownloads>

The objective of the solution is to automatically convert all C specifications into free format syntax, which includes the logic of the program and the more recent support for free format H, F, D, and P specifications. Although this task is straightforward for some operation codes (for example, **EVAL** and **ADD**), others are more complex (for example, **MOVE** and **MOVEL**). The choice of instruction to generate often depends on the types, lengths, and dimensions of fields that are used in Factor 1, Factor 2, or results factor.

For this reason, ARCAD-Transformer RPG uses the ARCAD repository metadata following a complete cross-reference analysis down to the field level to check the characteristics of each field in the program.

The converted source can be compiled directly, with user intervention required only for the handling of %Found and %Equal on **SCAN**, **CHECK**, **CHECKR**, and **LOOKUP**.

7.4 Preferred practices

The following sections describe some preferred practices for when you create modern RPG applications. These practices are divided into design practices and coding practices.

Some of these practices are similar to the ones in Chapter 2, “Programming RPG IV with Style”, in *Who Knew You Could Do That with RPG IV? A Sorcerer's Guide to System Access and More*, SG24-5402. The earlier book reprinted an article that is called “The Essential RPG IV Style Guide” by Bryan Meyers. RPG has changed much in the intervening years. The Style Guide is updated. Bryan Meyers' entire updated guide is not reprinted here, but you should look at it for his updated perspective. Many of the preferred practices in this section are similar to the ones in the updated guide. You can find the article at the following websites:

- ▶ <http://iprodeveloper.com/rpg-programming/rpg-iv-style-revisited>
- ▶ <http://www.bmeyers.net/faqs/14-tips/32-rpg-iv-style>

7.4.1 Preferred practices for RPG program design

Here is a list of the preferred practices for RPG program design:

- ▶ Usage of modular application design.

Modular designs make it easier to reuse components. Reusing components can be especially important for applications that have (or might have in the future) multiple application interfaces.

It is important to ensure that an application function works the same regardless of whether it was started from a “green screen”, a browser, or a mobile device. The best way to ensure that it works the same way is to call the same code from all interfaces. To do this, the logic must be written as separate callable functions. User interface logic should always be separated from background business logic, which should be separate from the data access layer.

- ▶ Use ILE to implement your modular design.

Service programs typically make the best home for application functions that are called from multiple programs. Binding directories simplify the usage of service programs. Binder language is needed to manage the usage of service programs throughout changes to the service program contents. For more information about these facilities, see 7.6, “ILE and modularization” on page 213.

- ▶ Centralize data declarations.

Use named constants instead of referring to constant values in the logic. Consider alphabetizing your declarations to make them easier to find. Never use compile-time arrays because the separation of the data from the definition makes them difficult to use and maintain. To initialize an array with constant data values, consider overlaying an array in a data structure that contains the values. Never use multiple occurrence data structures; instead, use data structure arrays, which are far more flexible.

- ▶ Use SQL and set-oriented processing where appropriate.

Using SQL for data access can often simplify and reduce the RPG logic that is required to accomplish a task. SQL also can use the latest improvements in DB performance and function. SQL is suited to processing sets of data in contrast to RPG's native record level access, in which program logic interacts with individual rows or records.

When you design programs, strive for the usage of set-oriented processing to use the advantages of SQL in situations where it can simplify the RPG logic that is needed. For more information about how to accomplish this task, see 8.3, “Accessing and operating on sets (SQL DML)” on page 236.

- ▶ Use RPG procedures instead of subroutines to structure code inside a module.

The ability to declare local data that can be used only in the procedure makes maintenance easier. In addition, function calls with parameters and return values provide far more information to developers looking at the code afterward than the name of the routine. This makes the code more easily understood and therefore more easily and reliably maintained. For more information about this subject, see *3 Good Reasons to Stop Writing Subroutines*, found at:

<http://www.ibmssystemsmag.com/ibmi/enewsletterexclusive/25782p1.aspx>

The usage of subprocedures requires the use of ILE, which is described in 7.6, “ILE and modularization” on page 213.

- ▶ RPG procedures should use local data whenever feasible.

Parameters that are being passed into procedures provide a more obvious data flow and tend to provide for more reliable maintenance than referring to global data inside a procedure.

There are occasions when sharing global data between procedures in a module might greatly simplify the logic, particularly when multiple procedures perform similar functions on the same set of data. Exceptional cases such as this one are why you should use local data “whenever feasible”. Those occasions should be the exception and not the rule for data access from procedures.

To avoid confusion, some shops use a naming convention, such as a “g_ prefix” on the name for any global variables and files. That naming convention makes it easier to spot global references within a subprocedure.

- ▶ Group large numbers of parameters together into one or more structures.

Structures also allow a procedure to return a number of values on a function call.

- ▶ Use prototypes for all calls.

Prototypes can be used for procedures, functions, and programs, including non-RPG programs.

When the call is likely to be done from multiple programs, externalize the prototype to a separate member that is included through `/Copy` or `/Include` at compile time. Multiple prototypes typically and normally are grouped into a single member for ease of management.

Beginning with IBM i 7.1, the compiler no longer requires prototypes in many situations. However, there are occasions when it is a preferred practice to include a prototype even if the compiler does not require it. If a subprocedure is not exported, no prototype is needed. Likewise, if an exported procedure is never called from RPG code (for example, if it is called only from CL), then a prototype is also not needed. Whenever RPG code is called from another RPG program or procedure, the prototype for the called code should be placed in a source member that is included (through `/Copy` or `/Include`) in both the calling and called source members because it allows the compiler to compare the Procedure Interface (PI) to the prototype to ensure that the prototype is coded correctly. Because the prototype member is included by the compiler in both the caller and called routine, this greatly reduces the potential for errors that are related to parameter passing.

For more information about using prototypes, see the following information:

- *Prototypes: For All Calls, All the Time*, found at:

<http://www.mcpressonline.com/programming/rpg/prototypes-for-all-calls-all-the-time.html>

- *Prototypes: Beyond CALL/PARM*, found at:

<http://iprodeveloper.com/rpg-programming/prototypes-beyond-callparm>

- ▶ Parameters should be declared as `CONST` (constant) if they are not updated by the called program or procedure.

This can greatly simplify the call logic because the caller can pass literals or expressions as parameters. In addition, no extra logic is required to accommodate small variations in parameter data definition, such as a packed field containing the data by the caller when the called routine expects a zoned decimal or an integer.

- ▶ Reduce cyclomatic complexity.

Code that is complex is harder to understand, maintain, and test. Modularity can be used to reduce complexity in some cases. Tools can be used to measure complexity in your code, along with other measurements that are useful in tracking your modernization efforts. To learn about reducing cyclomatic complexity and other metrics, and to learn about ideas on how to accomplish this task, see 7.9, “Cyclomatic complexity and other metrics” on page 218.

- ▶ Documentation.

Most programmers do not like documenting their code. With the usage of longer, more meaningful variable names and good comments, much of the code can be self-documenting to a great extent.

One common issue that occurs in shops that are writing reusable modules is how to document what functions exist in the shop so that developers do not waste time writing new code instead of calling an existing function. To solve this problem, some shops provide some extra documentation through comments in the members they use for the prototypes for the callable functions. The prototypes themselves provide much documentation for the function. With the addition of a few comments, there is often enough documentation for those functions for developers to use to see whether the function they want exists.

Some RPG programmers have gone a little further and have written code to extract the documentation from the prototype members and make it available in a more easily readable and better organized, perhaps searchable, format. Some examples of the code that has been written for this task can be found online. There are at least three different examples of this code that are publicly available for download. One example is called RPGLEDOC. It is documented on the IBM System i Developer downloads page:

<http://www.systemideveloper.com/downloadRPGLEDOC.html>

Another project that is called ILEDOCs is available for download from the following website:

<http://iledocs.sourceforge.net/index.php>

Another Sourceforge project is called RPGDOC and is available for download from the following website:

<http://rpgdoc.sourceforge.net/>

These examples might provide developers with a starting point or some ideas about how they can develop something similar or customized for their shop.

7.4.2 Preferred practices for RPG coding

This section describes some of the preferred practices for RPG coding:

- ▶ Avoid the *INnn numbered indicators in RPG logic.

Modern RPG logic should never (directly) refer to such indicators. Numbers are far less meaningful than variable names, so when using indicators, give them a name. Meaningful names help make code easier to read, understand, and maintain. Then, they reduce maintenance errors, making the code more reliable. Some developers choose to incorporate the indicator number into the name to provide an obvious connection to the DDS code.

The need to use numbered indicators was reduced in recent versions of RPG. For example, built-in functions are now used to determine I/O status results (such as %EOF and %FOUND) and error conditions (%ERROR).

There are still some occasions where numbered indicators must be used, most notably when communicating with DDS-defined screens or reports. However, there are several techniques that can be used to provide a more meaningful name to those indicators for use in the RPG logic. Here are three common techniques:

- Use an Indicator Data Structure (INDDS) to replace the numbered indicators in your program for display and printer files. An example of that technique is shown in Example 7-1. When using this technique, you must remember to always refer to the name when communicating to the display or printer file. Using the numbered indicators has no effect. Therefore, this technique tends to work better when writing new code than when retrofitting existing code to remove indicator references.

Example 7-1 Example using INDDS

```
FDispfile CF E WORKSTN INDDS(DspInd)

D DspInd DS
  // Function key response indicators
D Exit_F3 n Overlay(DspInd: 3)
D Return_F12 n Overlay(DspInd: 12)
  // Conditioning indicators
D DateError_30 n Overlay(DspInd: 30)
D StDateErr_31 n Overlay(DspInd: 31)
```

```

D  EndDateErr_32          n  Overlay(DspInd: 32)

D Today                  S          D  Inz(*Sys)

/FREE
  // Date validation logic
  StDateErr_31 = StartDate < Today;
  EndDateErr_32 = EndDate < StartDate;
  DateError_30 = StDateErr_31 Or EndDateErr_32;

  ExFmt TestRec;
  If Not Exit_F3 or Return_F12;
    // Process data from the screen
  Else;
    // End processing of this screen
  EndIf;

```

You can read more about the use of INDDS in the ILE RPG reference (http://www.ibm.com/support/knowledgecenter/ssw_ibm_i_71/welcome.html?lang=en) and in 6.2.7, “Indicator data structure”, of *Who Knew You Could Do That with RPG IV? A Sorcerer’s Guide to System Access and More*, SG24-5402. Example 7-1 on page 195 is similar to the example that is used in that publication with the logic converted to free format. The F and D specs are not converted to free format in these examples to help make it more understandable to those programmers that are not familiar with that style. A free format version of this code is included in 7.5.3, “Free format examples” on page 207.

- The second technique uses a named indicator as the index to the *IN array that contains the 99 numbered indicators, which exist in every RPG program automatically. The value of the named indicator is the numeric value of the indicator to be replaced. This technique is easier to use for retrofitting existing code. Whenever you find a numbered indicator and track down what its purpose is, create an appropriately named constant and use that with *IN to replace an old-fashioned direct reference to the numbered indicator. Although this is not good style, all other references directly to the number indicator in the program continue to work. An example is shown in Example 7-2.

*Example 7-2 Replacing numbered indicators using *IN array with named constants*

```

FDispfile CF  E          WORKSTN

  // Numbered indicator constants
D Exit_03      C          03
D Return_12   C          12
D DateError_30 C          30
D StDateErr_31 C          31
D EndDateErr_32 C          32

D Today        S          D  Inz(*Sys)

/FREE
  // Date validation logic
  *In(StDateErr_31) = StartDate < Today;
  *In(EndDateErr_32) = EndDate < StartDate;
  *In(DateError_30) = *In(StDateErr_31) Or *In(EndDateErr_32);

```

```

ExFmt TestRec;
If Not *In(Exit_03) or *In(Return_12);
  // Process data from the screen
Else;
  // End processing of this screen
EndIf;

```

-
- The third technique maps the *IN array to a data structure using a basing pointer. Technically, you can refer to either the renamed indicator subfield or the *INxx indicator when retrofitting existing logic. However, it is a preferred practice that all references to *INxx indicators be replaced by the renamed subfields to avoid confusing code that refers to the same indicator by two different names. This technique has the further advantage that the names in this case do not require the *IN prefix, so it might produce slightly more readable code. Some RPG users prefer to avoid the usage of pointers or might like the added information that a direct reference to *IN provides in the logic. An example is shown in Example 7-3.

*Example 7-3 Replacing numbered indicators mapping an *IN array with a pointer*

```

FDispfile CF E WORKSTN

D DspInd DS Based(pIndicators)
  // Function key response indicators
D Exit_03 n Overlay(DspInd: 3)
D Return_12 n Overlay(DspInd: 12)
  // Conditioning indicators
D DateError_30 n Overlay(DspInd: 30)
D StDateErr_31 n Overlay(DspInd: 31)
D EndDateErr_32 n Overlay(DspInd: 32)

D pIndicators S * Inz(%Addr(*In))

D Today S D Inz(*Sys)

/FREE
  // Date validation logic
StDateErr_31 = StartDate < Today;
EndDateErr_32 = EndDate < StartDate;
DateError_30 = StDateErr_31 Or EndDateErr_32;

ExFmt TestRec;
If Not Exit_03 or Return_12;
  // Process data from the screen
Else;
  // End processing of this screen
EndIf;

```

► Use free format syntax.

Free format syntax (compared to the use of fixed columnar format) enables far more readable code when it is used with appropriate indentation. As of IBM i 7.1 TR7, RPG H, F, D, and P specs can now be free format, as can logic statements. Because free format RPG is such an important topic, a separate section on coding in free format with examples is in 7.5, “Free format RPG” on page 204. This section describes free format coding in detail.

It is critical to use the appropriate indentation. When the code has statements to begin and end a block of code, the beginning and ending statements should appear at the same indentation level, and statements between the begin and end should be indented at least two spaces. This makes it simple to match up the beginning and ending of code blocks. Any blocks that are nested inside another block should be indented further. A simple example is included and shown in Example 7-4.

Example 7-4 Example of indentation in free format logic - before and after

* An example of columnar RPG logic - "Before"

```

C          READ(E)  TransFile
C          DOW      Not %EOF(TransFile)
C          IF       %Error
C          Eval     Msg = 'Read Failure'
C          LEAVE
C          ELSE
C      CustNo  CHAIN(N)  CustMast    CustData
C          Eval     CustName = %Xlate(UpperCase:
C                                     : LowerCase : CustName)
C          EXSR     CalcDividend
C          READ(E)  TransFile
C          ENDIF
C          ENDDO

```

// The same logic as above indented in free format - "After"

```

READ(E) TransFile;
DOW not %EOF(TransFile);
  IF %Error;
    Eval Msg = 'Read Failure';
    LEAVE;
  ELSE;
    CHAIN(N) CustNo CustMast CustData;
    Eval CustName = %Xlate(UpperCase : LowerCase : CustName);
    EXSR CalcDividend;
    READ(E) TransFile;
  ENDIF;
ENDDO;

```

- ▶ Manipulate character strings without arrays.

Modern RPG has many string handling operations and functions to make manipulation of text and character strings both simpler and more efficient. The usage of functions such as %SCAN, %REPLACE, and %SCANRPL can replace many lines of older style logic.

- ▶ Use RPG varying length character fields where appropriate.

Using varying length fields can be more efficient and simplify logic. For example, when concatenating text repetitively into a single field, defining the receiving field as varying avoids the need to trim trailing spaces from it before each iterative concatenation and avoids subsequent padding of the receiving field with blanks. Although the difference in logic is subtle, the resulting efficiency of the code can be substantial. It is necessary to trim the field that is being added unless it is also a varying field.

Example 7-5 on page 199 shows an example of the difference. The code builds a string of 100 comma-separated values, first into a "normal" fixed-length field and then into a varying length field. The second example is much more efficient.

Example 7-5 RPG varying length character fields

```
D fixedArray      S           10A   Dim(100)
D longFixed       S           1100A
D longVarying     S           1100A   Varying
D i               S              3I 0

      // Fixed length Receiver
      // Each iteration trims longFixed, then pads with blanks
For i = 1 to 100;
    longFixed = %TrimR(longFixed) + ',' + fixedArray(i);
EndFor;

      // Varying length Receiver
      // Each iteration trims shorter fixArray(i) - no padding
For i = 1 to 100;
    longVarying += ',' + %TrimR(fixedArray(i));
EndFor;
```

In addition to the efficiencies of varying length fields in repetitive string manipulation, they are also effective when passed as parameters. Instead of requiring a second parameter to specify the length of the value that is passed (such as is required when using QCMDEXC), a varying length parameter carries its own length with it. For procedures where that information is important, it is a far easier and more efficient way to do it.

For more information and examples of using varying length fields, see the following articles:

- *The Efficiency of Varying Length Character Variables*, found at:
<http://www.itjungle.com/fhg/fhg091008-story01.html> and
- *The Versatility of Variable-Length Fields*, found at:
<http://www.ibmssystemsmag.com/ibmi/developer/rpg/iSeries-EXTRA--The-Versatility-of-Variable-Length/>

- Use ALIAS names for tables or files that have them.

With the increased usage of SQL DDL to create database tables or views, many developers are giving longer, more meaningful names to columns. Given the six-character name size limitation for older versions of RPG, databases that were created many years ago typically have cryptic column names that can make code using those names difficult to understand. When re-creating these tables with SQL, an alternative, more descriptive SQL name is often supplied. Some shops might have alternative names even if they used DDS to create the files by using the **ALIAS DDS** keyword. Even shops that are not planning to convert their existing databases to use DDL might want to consider adding more meaningful DDS ALIAS names to make their logic using those files more readable. To use these better names in your RPG programs, use the **ALIAS** keyword.

The **ALIAS** keyword can be specified on an F spec for a file that uses Data Structure I/O with a data structure using the **LIKEREC** keyword. It can also be used with an externally described data structure. In both cases, the subfield names that are brought into the data structure by the compiler are the alias or SQL names. Fields (or columns) without separate alias or SQL names come into the structure with the standard name or system name.

All native (meaning non-SQL) file I/O statements must use the data structures to replace the shorter standard names with the alternative names. The compiler does not accept the **ALIAS** keyword for a file that generates Input or Output specs.

- ▶ Use comments appropriately to document your code.

The usage of modern logic constructs and meaningful variable names help make code more understandable and self-documenting. However, there is still a need for developer comments in the code to make the code clearer. Ensure that your comments elaborate on the code instead of merely repeating it. The usage of free format code allows for brief comments to be added at the end of lines of code so that comments are not required to always occupy an entire line by themselves.

- ▶ Avoid obsolete code.

One of the benefits of coding in free format is that to some extent it forces developers to use modern techniques. For example, operation codes that are replaced with built-in functions are, usually, not allowed in free format RPG logic. Likewise, the resulting indicators and defining variables in the logic are not supported.

Even if you are not coding your logic in free format yet, you should still use modern coding practices. If you are using free format, use the most modern coding techniques that are supported in the release that you are using. Typically, the more modern techniques make your code clearer, and therefore more maintainable, and are often more efficient than the older techniques.

- ▶ Use well-designed templates for new programs

Most developers have not started a new program from a blank screen since they were first learning RPG. Too often, however, the code that is copied as a starting point for the new inquiry program or the new subfile program is not a template that is designed for that purpose. Choosing a program that does something similar to what you need to do is not a good choice.

Every shop should have a set of templates for commonly used application functions. These templates should embody good style principles, such as those outlined here, and have easily identifiable spots in the code where customization is required for the specifics of the new program's function. Although it takes time to develop such a library of program templates, the payback over time can be dramatic. It becomes faster to customize a template program than to redesign a program that is designed for a different purpose to fit a new purpose. More importantly, the usage of modern templates helps ensure that newer coding styles and techniques are used more easily. Standard templates can also help enforce a standard coding style for your shop.

You might want to begin your template development process by searching for existing template programs that are shared by fellow RPG users on the internet and in books and articles. However, it is always wise to review the coding techniques in templates carefully to ensure the style and function of the code fits your shop's requirements. The template programs serve as the foundation for most, if not all, new programs in the future, so make sure that you are not replicating code with bad practices. It is also important to remember to update your templates regularly to keep up to date with the latest capabilities.

- ▶ Use appropriate, consistent, and modern error handling techniques.

Too many RPG programs have inadequate error handling in the code. In these cases, the default RPG error handling mechanisms are used and application users are faced with a decision about how to respond to system error messages. Relying on default error handling, especially as you move into using features of the ILE, might cause inconsistent application behavior because ILE has different default error handling than the Original Program Model (OPM).

This topic has many facets and many possible solutions. The modern error handling challenge and some suggested solutions are covered in *RPG: Exception and Error Handling*, REDP-4321.

- ▶ Identify and remove “dead code”.

Many older applications contain code that is no longer in use, which is typically because of a change in the business and application rules logic that was replaced, but the old “dead” logic remains in place. Although it might seem that code that is not running cannot present much of a problem, it can contain hidden risks.

Identifying dead code is difficult and few tools seem to exist to help. One company has created some tools to help in their own challenge to uncover dead code and has made the source code for the tool they used available to the RPG community. Section 7.8, “Dead code elimination” on page 215 covers the issue of dead code and the tools that might provide some help in dealing with it.

- ▶ Use qualified data structures and qualified files where appropriate.

RPG developers have come a long way since the old days of language-enforced 6-character names. Longer variable names, naming numbered indicators, and the usage of longer alias names all contribute to making code more understandable. In some cases, the usage of qualified names can make similar enhancements to code clarity.

The keyword **QUALIFIED** can be used on data structures to allow qualification of subfield names. For example, a qualified data structure that is named `MailingAddress` might contain a subfield that is named `Street`. In the RPG logic, that subfield is referred to as `MailingAddress.Street`. This makes the relationship between the structure and the subfield more obvious and allows for a second structure that is named `BillingAddress` to also use the subfield name `Street` while holding different data.

A File spec might also contain the **QUALIFIED** keyword to allow for the usage of qualified record format names in the logic.

- ▶ Use data structures for I/O where appropriate.

When using native (meaning non-SQL) I/O, using a data structure to receive the data on input or to be the source of the data for output can help make some logic more understandable. The usage of qualified structures also provides documentation for the source of the data names.

Because using structures for I/O means the I and O specifications for the file are likely not generated, the usage of **EVAL-CORR** (Eval Corresponding) between structures can simplify movement of data between structures.

The usage of structures for I/O is required in some cases. Local files in subprocedures must always use structures for I/O. The use of more meaningful **ALIAS** names also requires the use of structures for I/O operations.

- ▶ Use local files and file parameters where appropriate.

Local files can be declared in subprocedures as of Version 6.1. Local files must use data structures for I/O and can use the **STATIC** keyword to avoid the impact of closing and reopening the file on each call to the subprocedure.

In many cases, the ability to pass a file as a parameter to another subprocedure or program can be even more useful. This has a similar effect to the old shared open data path techniques that RPG users have used for a while. However, because shared open data paths are implemented through CL and not in the RPG program, the developer has far less control over the usage of the file and how many other programs have access to the shared data. It was often a source of errors that were introduced into applications during maintenance. When a file is passed as a parameter, the sharing is done only between the caller and the called code. This makes for a far safer mechanism for sharing as well as being obvious to developers maintaining the code. This allows the caller to “own” the file and to pass it to the called routine, who can access the data at the current file cursor position, and potentially update the data independently and reposition the file's cursor.

Specifying the **LIKEFILE** keyword for the parameter in the prototype for the called routine establishes the parameter to be a file. In the called routine, the F spec for the file that is received as a parameter uses the **TEMPLATE** keyword and potentially also the **BLOCK** keyword because it is important that the called routine know what level of blocking, if any, for the file is in use in the caller.

For more information about these topics, see the following articles:

- *Major Changes in RPG File Handling*, found at:

<http://www.ibmssystemsmag.com/ibmi/developer/rpg/Major-Changes-in-RPG-File-Handling/>

- *Files in Subprocedures*, found at:

<http://www.itjungle.com/fhg/fhg042810-story01.html>

Sharing in V6, found at:

<http://www.itjungle.com/fhg/fhg032112-story01.html>

- ▶ Create a Linear Main program when the RPG cycle is not needed.

Most RPG developers have not used any part of the RPG cycle (aside from *INLR) for many years. However, by default, the cycle logic is still compiled into every RPG program that is written, even if it is not used. To avoid the extra impact of the unnecessary RPG cycle logic, developers can code the main procedure for the program (or module) as a subprocedure and use the **MAIN** keyword on the H (Control) specification to identify it as the main procedure.

Many RPG users have developed reusable subprocedures in modules that go into service programs. Those modules typically have the **NOMAIN** Control specification keyword specified. That setting lets you avoid the RPG cycle logic, but then that module cannot be used as an entry module in a program. The **MAIN** keyword allows a subprocedure to play the role of a main procedure so that the module can be an entry module.

For more details about the differences between the original style of RPG module, now called Cycle-Main, compared to the new Subprocedure-as-Main-procedure, called Linear-Main modules, see Chapter 3, “Procedures and the Program Logic Cycle”, of the *RPG Reference* manual, found at:

<https://pic.dhe.ibm.com/infocenter/iseries/v7r1m0/index.jsp>

- ▶ Take advantage of information from the compiler.

Many RPG developers are not aware of some valuable information that the compiler puts in the compiler listing. Many modern developers are using Rational Developer for i and because the compile error portion of the compiler listing is not needed by them, they rarely look at compiler listings. However, there are some cases where the compiler listing might still hold some valuable information.

For example, when the **EVAL-CORR** operation is used, the compiler listing provides details of how each subfield is handled. If you are concerned about potential loss of data because of CCSID conversions in your programs, you can use the **CCSIDCVT(*LIST)** option on the H spec to include details of all CCSID conversions and warnings when any of them might cause loss of data.

In addition, some developers find the field cross-reference in the compiler listing helpful for showing where global variables are being used in subprocedures. Rational Developer for i users can also get similar cross-reference information from the Outline View.

7.4.3 Quick list of RPG preferred practices guides

This section lists some RPG preferred practices guides:

- ▶ *Who Knew You Could Do That with RPG IV? A Sorcerer's Guide to System Access and More*, SG24-5402
- ▶ Brian Meyers
 - *The Essential RPG IV Style Guide*, found at:
<http://www.bmeyers.net/faqs/14-tips/32-rpg-iv-style>
 - *RPG IV Style Revisited*, found at:
<http://iprodeveloper.com/rpg-programming/rpg-iv-style-revisited>
- ▶ Other articles
 - *3 Good Reasons to Stop Writing Subroutines*, found at:
<http://www.ibmssystemsmag.com/ibmi/enewsletterexclusive/25782p1.aspx>
 - *Prototypes: For All Calls, All the Time*, found at:
<http://www.mcpressonline.com/programming/rpg/prototypes-for-all-calls-all-the-time.html>
 - *iPrototypes: Beyond CALL/PARM*, found at:
<http://iprodeveloper.com/rpg-programming/prototypes-beyond-callparm>
 - *RPG: Exception and Error Handling*, REDP-4321
 - *Major Changes in RPG File Handling*, found at:
<http://www.ibmssystemsmag.com/ibmi/developer/rpg/Major-Changes-in-RPG-File-Handling/>
 - *Files in Subprocedures*, found at:
<http://www.itjungle.com/fhg/fhg042810-story01.html>
 - *Sharing in V6*, found at:
<http://www.itjungle.com/fhg/fhg032112-story01.html>
 - *Free-Ride--A-look-at-free-form*, found at:
<http://www.ibmssystemsmag.com/ibmi/developer/rpg/iSeries-EXTRA----Free-Ride--A-look-at-free-form-ca/>
 - Martin, *Free-Format RPG IV, 2nd ed.*, MC Press, 2013. ISBN 1583473475
- ▶ ILE Details by Scott Klement
 - *ILE Basics: It's All About the Call*, found at:
<http://iprodeveloper.com/rpg-programming/ile-basics-its-all-about-call>
 - *ILE Basics: Procedures*, found at:
<http://iprodeveloper.com/rpg-programming/ile-basics-procedures>
 - *ILE Basics: Modules and Programs*, found at:
<http://iprodeveloper.com/rpg-programming/ile-basics-modules-and-programs>
 - *A Beginner's Tour of Activation Groups*, found at:
<http://iprodeveloper.com/rpg-programming/beginners-tour-activation-groups>

7.5 Free format RPG

Free format syntax is an important aspect of modern RPG applications. Although free format logic has been available for years, it is relatively recently that the ability to write nearly all RPG code in free format was introduced.

This section describes the advantages of coding with free format syntax. We also take a closer look at the syntax differences relating to RPG IV syntax and how that syntax has changed in the years since RPG IV was introduced.

7.5.1 Why free format

There are many practical reasons to use free format RPG syntax. The following list provides some of the significant reasons for why free format syntax is at the heart of modern RPG applications.

- ▶ Indented logic is more understandable.

Free format gives the developer the opportunity to indent statements to show the structure of the code. The code is more easily understood, especially in cases involving complicated nested logic blocks, such as nested IF statements, Select/When statements, or Monitor blocks. The easier the code is to understand, the faster, less error prone, and maintainable it is.

- ▶ More efficient usage of space in source code.

This is especially true in the logic of the program. Before free format RPG logic was available, many developers found that they rarely used Factor 1. So, large parts of the left side of the C spec were blank. At the same time, the usage of expressions using built-in functions might get lengthy, often requiring multiple lines because only the relatively small Factor 2 area could be used.

With free format logic, the logic can start further to the left, leaving much more room for complex expressions. The edit screen is cleaner when the space is used better.

- ▶ More room for larger names.

Longer variable and procedure names and qualified names are more descriptive and can make code more understandable. Free format logic means that there are no more limitations on the size of the names that you use, unless you consider the 4096 character limit for RPG names to be restrictive.

- ▶ New features are only available in free format.

There are a few language functions that are only available in free format logic. Two examples of this are the new ways to express keys for operations such as **CHAIN** or **SETLL**. One way is to embed the variables to be used to match the key values into the statement in Factor 1, typically in parentheses, after the operation code. The other way is to use the %KDS (Key Data Structure) built-in function.

- ▶ Works well with the new modern development tools.

Free format works well with modern editors such as Rational Developer for i. This tool helps you with indenting and formatting of the free format code. Using tools such as the outline view and content assist can greatly improve the accuracy and productivity of a developer.

- ▶ Attraction of new RPG programmers.

Free format logic brings RPG more inline with other modern programming languages, all of which use a free format syntax. This is important for attracting new developers who are coming into the marketplace. Traditional, fixed-format RPG gives RPG the undeserved appearance of an old-fashioned language that is not up to modern application tasks. RPG is a powerful and flexible language that many developers come to prefer over other language options for business applications. However, they are not likely to be attracted to the older format of columnar RPG.

7.5.2 Free format RPG syntax overview

Before Version 5 Release 1, all RPG logic and declarations were coded in strict columnar format. This is the traditional RPG that was inherited from earlier versions to the new RPG IV language. An example is shown in Example 7-7 on page 207. This section provides three examples of the same program.

The ability to code RPG logic in free format started with Version 5 Release 1. This was a boon to developers who could finally indent their nested logic to make it easier to understand. This limited free format syntax required the usage of the `/Free` directive to go into free format and the `/End-Free` directive to leave free format. RPG programmers using modern code techniques such as subprocedures were required to code many `/Free` and `/End-Free` directives. In addition, the language still required strict columnar format for declaring files and other data and beginning and ending procedures.

Example 7-8 on page 209 shows the same example as Example 7-7 on page 207, but it is coded in the limited free format syntax that was available from Version 5 Release 1 through Version 6 Release 1. The operation code begins the statement, followed by the two factors and the result, and each statement ends with a semi-colon (;). There are two operation codes that can be omitted: **Eva1** and **Ca11P**. Many operation codes that were supported in the fixed format cannot be used at all in free format logic, which allowed the language to begin to effectively deprecate many of the traditional coding practices from its early years. For more information about the rules of this free format logic syntax, see *iSeries EXTRA -- Free Ride: A look at free-form calc support*, found at:

<http://www.ibmssystemsmag.com/ibmi/developer/rpg/iSeries-EXTRA----Free-Ride--A-look-at-free-form-ca/>

Free-Format RPG IV, Second Edition by Martin also contains information about this topic.

Before IBM i 7.1 TR7, only the logic portion of an RPG program could be written in free format. As the example in Example 7-8 on page 209 shows, the `/Free` and `/End-Free` compiler directives were required and the interruption of the free format logic with fixed-format P, F, and D specs create cumbersome coding when using subprocedures.

Beginning with IBM i 7.1 TR7, many syntax rules were relaxed. Most spec types, certainly the most commonly used spec types, can now be coded with the only concern for columns being that the code must fall between columns 8 - 80. Specifically, H, F, D, and P specs became free format. Furthermore, it was no longer necessary to use the compiler directives `/Free` and `/End-Free`. If there are blanks in columns 6 and 7 of a statement, the compiler treats the code as free format. Developers can now intermix free format operations with fixed format (columnar) operations without the usage of any compiler directive. Any `/Free` and `/End-Free` directives that are included in the code are ignored by the compiler.

The I and O specifications still require columnar format. Likewise, code that is related to the RPG cycle requires columnar format. These are older parts of the RPG IV language that are not widely used in modern applications.

Replacing the spec type is an operation code (also known as an opcode) that is based on the type of spec that is being replaced. To replace an H spec, the opcode is **CTL-0PT**. To replace the F, D, or P specs, it is a **DCL-xx** (DeCLare) operation code, where “xx” represents the type of declaration (formerly in columns 24 - 25 of the D spec). In the list of **DCL-xx** operations that is shown in Figure 7-1 on page 217, unless otherwise specified, the spec type that is being replaced is D. The opcodes that are shown in Table 7-1 begin the declaration and there is a corresponding ending opcode that is required for most of them, such as **END-DS**, **END-PR**, and **END-PROC**.

Table 7-1 Operation codes

Opcode	Declaration for
DCL-F	A file (replacing F spec)
DCL-S	A stand-alone field
DCL-DS	A data structure
DCL-C	A named constant
DCL-PR	A prototype
DCL-PI	A procedure interface
DCL-PROC	A procedure (replacing the P spec)

Developers can now intermix file declarations with other data declarations. This is even true for the fixed-format F and D specs. This has the advantage of allowing coding of the file declaration with the data structure populated from that file together. Another example is shown in Example 7-9 on page 211, where the overflow indicator variable is coded directly following the printer file declaration.

The move toward using keywords to replace esoteric columnar based codes that started with the original RPG IV D-specs has been extended to file specifications. Keywords now replace all the non-intuitive columnar syntax of the fixed-format F spec. The new **DCL-F** operation works only with full procedural files. Some well-chosen default values allow the omission of many other parts of the F spec syntax. For example, **DISK** is the default type for the new **DEVICE** keyword as is “externally described”. The following declaration is for an externally described customer master database file that is open for input only:

```
Dcl-F CustMast;
```

The **USAGE** keyword is now used to specify the file's capabilities through the values ***UPDATE**, ***INPUT**, ***OUTPUT**, and ***DELETE**. The default usage varies by **DEVICE** type, for example, **DEVICE(PRINTER)** files are **USAGE(*OUTPUT)** by default and **DEVICE(WORKSTN)** files default to **USAGE(*INPUT: *OUTPUT)**. The old **K** in the F spec is replaced by the keyword **KEYED**. For an example of declaring a file that requires the **USAGE** keyword, see Example 7-9 on page 211.

Because the D spec already used more keywords than the F spec did, its replacement syntax has not changed as dramatically. However, there are many new keywords to replace the old columnar definitions. Table 7-1 listed that the definition types (for example, **DS**, **S**, **PR**, and **PI**) were replaced by the type of declaration opcode that is used (**DCL-DS**, **DCL-S**, **DCL-PR**, and **DCL-PI**). In addition, the data type is also keyword-based. Fixed-length alphanumeric fields are now declared as **CHAR(len)** and packed decimal fields are **PACKED(digits:decimals)**. When defining packed decimal fields, the default is for 0 decimals so the second part of the parameter is often not needed. The format of Date fields is now part of the data type declaration, rather than a separate keyword. The name of the thing being declared must be first in the **DCL-xx** statement. For cases where a name is not needed, the special value ***N** can be used instead. The use of ***N** is illustrated in Example 7-9 on page 211.

Here are a few examples of some stand-alone field declarations:

```
Dcl-S CustName Char(10);
Dcl-S AcctBalance Packed(5:2);
Dcl-S Count Packed(3);
Dcl-S AnotherCount Int(3);
Dcl-S OrderDate Date(*MDY);
```

Data structures have an opcode for a subfield, **DCL-SUBF**. However, like **EVAL** and **CALLP**, that opcode is optional in most cases. The exception is when a subfield is the same as an op code name, such as **SELECT** or **READ**. Although most of the changes for free format have relaxed rules, there is at least one thing that can be done in fixed-format D specs that cannot be done in the new free format declaration specs. The **OVERLAY** keyword cannot reference a data structure. It can reference only a subfield in a free form DS declaration. However, the new **POS** (position) keyword can be used to accomplish the same effect by specifying **POS(1)** to overlay on the first position of the DS. The **POS** keyword can also be used to replace the usage of the **From** column in the old D specification.

A simple data structure might look something like Example 7-6. Note the ability to indent nested or redefined subfields.

Example 7-6 Data structures

```
Dcl-DS Address;
  Street Char(30);
  City Char(20);
  State Char(2);
  ZipCode Zoned(9);
    Zip Zoned(5) Overlay(ZipCode);
    ZipPlus Zoned(4) Overlay(ZipCode:5);
End-DS Address;
```

7.5.3 Free format examples

This section has examples of the same RPG IV program in three different stages of syntactical modernization to illustrate the differences and advantages of using the more modern free form syntax.

Example 7-7 shows the fixed-format example. That is an example of how code was written before Version 5 Release 1. Even though it is in columnar format, it still uses many of the other preferred practices of modern RPG, such as using subprocedures with local data in place of subroutines. It also uses built-in functions in place of obsolete operation codes, such as **MVR** and **SUBDUR**. The code also uses blank lines to help break up the logic blocks, which does help make the nested logic blocks more readable. The program is relatively modern in style except for the columnar syntax. Even though this program does not use complex expressions, it was still necessary to use two lines for some expressions because of the unusable space that is left for Factor 1 and the conditioning indicator.

Example 7-7 Original fixed-format RPG example

```
*-----
* Source:      DateCalcs1
*
* Description: Simple Fixed Form RPG program
*              with 2 internal subprocedures
*-----
```

```

H DftActGrp(*No)

FDateErrRpt0 E PrinterOFLIND(newPage)
FDateFile UF E Disk

* Overflow indicator - init to *on to force heading on 1st page
D newPage S n Inz(*On)

C Read DateFile
C DoU %EOF(DateFile)
C Eval Count+= 1

C *CYMD Test(DE) CYMD

C If %Error
C If newPage
C Eval pageNum += 1
C Write Heading
C Eval newPage = *Off
C EndIf

C Write Detail

C Else

* Update record with additional data
C Eval fullDate=%Date(CYMD:*CYMD)
C Eval dayName=NameOfDay(fullDate)
C Eval dayNumber=DayOfWeek(fullDate)
C Update DateRec

C EndIf

C Read DateFile

C EndDo

* User pressed F3 so set LR and exit

C Eval *InLR = *On

* DayOfWeek - Calculates day of week according to ISO standard
* i.e. Monday = 1, Tuesday = 2, etc.
* - Input: WorkDate (Date field in *ISO format)
* - Return: Single digit numeric

P DayOfWeek B

D workDate PI 1S 0
D workDate D Const DatFmt(*ISO)

* anySunday can be set to the date of ANY valid Sunday

D anySunday S D INZ(D'2013-10-27')
D workNum S 7 0

```

```

D workDay          S          1S 0

C          Eval      workDay = %Rem(
C          %Diff(workDate: anySunday: *D) : 7)
C          If        workDay < 1
C          Eval      workDay += 7
C          EndIf

C          Return    workDay

P DayOfWeek        E

* NameOfDay - Calculates alpha day name for any date
*           - Uses:   DayOfWeek
*           - Input:  WorkDate (Date field in *ISO format)
*           - Return: Alpha day name (9 characters)

P NameOfDay        B

D          PI          9
D workDate          D   Const DatFmt(*ISO)

D          DS
D dayData           42   Inz('Mon  Tues  Wednes+
D                   Thurs Fri   Satur Sun  ')
D dayArray          6   Overlay(dayData) Dim(7)

C          Return     %TrimR( dayArray( -
C                   dayOfWeek(workDate) ) ) + 'day'

P NameOfDay        E

```

The next iteration of this code is shown in Example 7-8. This example represents how the code can be written beginning with Version 5 Release 1 by using the original free format logic. This makes the logic portion of the code more understandable, with indentation making the logic flow of the nested IF blocks much more obvious. However, the usage of subprocedures required the code to go back and forth between columnar format and free format and required the use of /Free and /End-Free.

This example is written assuming that the code must be compiled on a release before IBM i 7.1 TR7. For Version 7.1 and later, the /Free and /End-Free directives can be removed from this example, even without using the new syntax for declaring data and files and for beginning and ending procedures.

Example 7-8 Free format logic example using Version 5 Release 1 support

```

//-----
// Source:      DateCalcs2
//
// Description: Converted to use V5R1 /Free syntax for calcs
//-----

H DftActGrp(*No)

FDateErrRpt0     E          Printer OFLIND(newPage)

```

```

FDateFile UF E Disk

// Overflow indicator- Init to *on to force heading on 1st page
D newPage S n Inz(*On)

/Free
Read DateFile;

DoU %EOF(DateFile);

Count+= 1;

Test(DE) *CYMD CYMD;

If %Error;
If newPage;
pageNum += 1;
Write Heading;
newPage = *Off;
EndIf;

Write Detail;

Else;

// Update record with additional data
fullDate = %Date(CYMD: *CYMD);
dayName = NameOfDay(fullDate);
dayNumber = DayOfWeek(fullDate);
Update DateRec;

EndIf;

Read DateFile;

EndDo;

*InLR = *On;
Return;

// DayOfWeek - Calculates day of week according to ISO standard
// i.e. Monday = 1, Tuesday = 2, etc.
// - Input: workDate (Date field in *ISO format)
// - Return: Single digit numeric

/End-Free

P DayOfWeek B

D workDate PI 1S 0
D workDate D Const DatFmt(*ISO)

// AnySunday can be set to the date of ANY valid Sunday

```



```

D anySunday      S          D   Inz(D'2013-10-27')
D workNum        S          7  0
D workDay        S          1S 0

/Free

workDay = %Rem( %Diff( workDate: anySunday: *D) : 7 );

If workDay < 1;
  workDay += 7;
EndIf;

Return workDay;

/End-Free
P DayOfWeek      E

// NameOfDay - Calculates alpha day name for any date
//           - Uses:   DayOfWeek
//           - Input:  workDate (Date field in *ISO format)
//           - Return: Alpha day name (9 characters)

P NameOfDay      B

D               PI          9
D workDate      D          Const DatFmt(*ISO)

D               DS
D dayData       42         Inz('Mon  Tues  Wednes+
D               Thurs Fri   Satur Sun  '
D dayArray      6         Overlay(dayData) Dim(7)

/Free

Return %TrimR( dayArray( DayOfWeek(workDate) ) ) + 'day';

/End-Free
P NameOfDay      E

```

Finally, Example 7-9 shows the new fully free-form version of the program. Not only does the code look more elegant without the need for all the columnar declarations for procedures and data, the newer support also allows for the placement of variable definitions that are related to files to be placed next to the file definition. In this example, you can see this placement with the declaration of the overflow indicator for the printer file.

Example 7-9 Free format RPG Example - after IBM i 7.1 TR7

```

//-----
// Source:      DateCalcs3
//
// Description: Modified to use V7 TR7 Full free form syntax
//-----

Ctl-Opt DftActGrp(*No);

```

```

Dcl-F DateErrRpt Printer OFLIND(newPage);
  // Overflow indicator- init to *on to force heading on 1st page
Dcl-S newPage ind Inz(*On);

Dcl-F DateFile Usage(*Update);

Read DateFile;

DoU %EOF(DateFile);

  Count+= 1;

  Test(DE) *CYMD CYMD;

  If %Error;
    If newPage;
      pageNum += 1;
      Write Heading;
      newPage = *Off;
    EndIf;

    Write Detail;

  Else;

    // Update record with additional data
    fullDate = %Date(CYMD: *CYMD);
    dayName = NameOfDay(fullDate);
    dayNumber = DayOfWeek(fullDate);
    Update DateRec;

  EndIf;

  Read DateFile;

EndDo;

*InLR = *On;
Return;

// DayOfWeek - Calculates day of week according to ISO standard
//               i.e. Monday = 1, Tuesday = 2, etc.
//               - Input: workDate (Date field in *ISO format)
//               - Return: Single digit numeric
Dcl-Proc DayOfWeek;
  Dcl-PI *N Zoned(1);
  workDate Date(*ISO) Const;
  End-PI;

  // AnySunday can be set to the date of ANY valid Sunday
Dcl-S anySunday Date(*ISO) Inz(D'2013-10-27');
Dcl-S workNum Packed(7);
Dcl-S workDay Zoned(1);

workDay = %Rem( %Diff( workDate: anySunday: *D) : 7 );

```

```

If workDay < 1;
    workDay += 7;
EndIf;

Return workDay;

End-Proc;

// NameOfDay - Calculates alpha day name for any date
//           - Uses:   DayOfWeek
//           - Input:  workDate (Date field in *ISO format)
//           - Return: Alpha day name (9 characters)
Dcl-Proc NameOfDay;
    Dcl-PI *N Char(9);
        workDate Date(*ISO);
    End-PI;

Dcl-DS *N;
    dayData Char(42)
        Inz('Mon  Tues  WednesThurs  Fri  Satur  Sun  ');

    // Redefine dayData as 7 x 6 array of names
    dayArray Char(6) Dim(7) Overlay(dayData);
End-DS;

Return %TrimR( dayArray( DayOfWeek(workDate) ) ) + 'day';

End-Proc;

```

7.6 ILE and modularization

RPG IV can be used as an ILE language or as a non-ILE language. When the **DFTACTGRP(*YES)** (Default Activation Group) parameter value is specified (or used by default) with the **CRTBNDRPG** command, the program is created in a mode that is more compatible with non-ILE programs. This non-ILE mode, however, creates programs that cannot use the ILE facilities that make modularization simpler and more efficient. Creating programs using either the **CRTPGM** command (after a **CRTRPGMOD** command) or the **CRTBNDRPG** command with **DFTACTGRP(*NO)** creates a program that can take full advantage of ILE facilities.

The primary advantage of ILE is modularization. Writing code in small, reusable pieces is recognized throughout the IT industry as a good way to enhance developer productivity, application reliability, and flexibility of the code for future usage.

If a developer can take advantage of pre-created functions or modules, it means that less new code must be written. Even more significant, however, is the impact on testing time and effort that is required. Because smaller individual functions can be more easily, thoroughly, and independently tested, applications that are written using those functions tend to have fewer errors. Even if thorough unit testing of each function is not done, the simple fact that the function is working well in other application scenarios means that the function has been tested more thoroughly than newly written code.

Writing in smaller “application-function-oriented” pieces also helps to prepare the application for the future. It is important to separate any code that is related specifically to the user interface because it is certain that user interface options change over time. Developers should strive to modularize the application function so that the function can be used in any user interface scenario. For example, functions such as **GetCustomerInfo** or **GetProductPrice** can be used as effectively in green screen applications as in web or mobile applications. They can be used as user-defined functions or stored procedures that are called from SQL or they can be called more directly through procedure calls from application servers or other environments. They are also likely to be as effective for future user or system interface requirements.

ILE provides several options to facilitate modular coding. RPG subprocedures are typically used to write the callable functions. The usage of subprocedures, even internal ones that are used instead of subroutines, requires ILE. Subprocedures are compiled into RPG MODULE objects. A module can contain a single function or similar functions can be grouped within a single module.

Module objects that contain functions that are destined to be called from multiple programs are grouped into Service Program (*SRVPGM) objects. These service programs are the ideal place to store code that is called by several programs because they do not require making a copy of the code for each program that uses the module. This makes the applications smaller and more efficient and means that they are more easily maintained because there is only one copy of every module to be replaced when updates occur.

There is more to ILE than can be described here, but there are many excellent sources that are available if you want to learn more about it. One important aspect of ILE that is not addressed here is the use of Activation Groups. Chapter 4, “An ILE guide for the RPG programmer”, in *Who Knew You Could Do That with RPG IV? A Sorcerer's Guide to System Access and More*, SG24-5402 is a good resource about ILE. This book was written in 2000, so some of the details might be out of date, but it can still form a good foundation for studying the subject.

Another excellent source of information can be found in a series of articles on ILE Basics that were written by Scott Klement and published on the iProDeveloper website. Here are those articles:

- ▶ *ILE Basics: It's All About the Call*, found at:
<http://iprodeveloper.com/rpg-programming/ile-basics-its-all-about-call>
- ▶ *ILE Basics: Procedures*, found at:
<http://iprodeveloper.com/rpg-programming/ile-basics-procedures>
- ▶ *ILE Basics: Modules and Programs*, found at:
<http://iprodeveloper.com/rpg-programming/ile-basics-modules-and-programs>
- ▶ *A Beginner's Tour of Activation Groups*, found at:
<http://iprodeveloper.com/rpg-programming/beginners-tour-activation-groups>

7.7 A modern language deserves modern tools

Many RPG users modernize their usage of the RPG language but are still using the traditional PDM/SEU toolset to edit their code. The advantages of the Eclipse-based Rational tools are even more obvious in a modern, modular application. One example is the ability to use the Outline view and Content assist to get information about callable functions and their parameters without needing to look at the source or even the prototype code for the called function. Another example is the simplicity that is afforded by having multiple source members open at once and even visible at once because of the flexibility of splitting the edit screen multiple times in any combination of horizontal and vertical lines.

Perhaps most importantly, the PDM/SEU editor is not updated to handle the latest versions of RPG syntax, including (but not limited to) the support for free format replacements for H, D, and P specs in IBM i 7.1. The new language features have not been updated since IBM i 6.1. As a result, it becomes more difficult to continue using the old editing tools while using modern facilities in the RPG language as more new features become available.

For more information about the Eclipse-based Rational tools that fully support the most current versions of RPG, see Chapter 4, “Modern development tools” on page 45.

7.8 Dead code elimination

Dead code is a phenomenon that affects programs independently from the programming language, paradigm, and platform. This section introduces the concept of dead code and how to detect it.

7.8.1 Why code “dies”

The term “dead code” refers to source code that exists in a program but it is not being run in the production environment. But, why does it “die”? This phenomenon usually happens because of a change in business rules. The business that is supported by the application is highly dynamic. Then, many business rules that are embedded in the code can become obsolete with time. Parts of the code are enforcing some business rules that are not used anymore, so the code probably is not going to run.

Dead code sometimes is not detected and deleted for the following reasons:

- ▶ Lack of application understanding

Programmers who are not familiar with the code are more likely to keep code intact, even if it seems to be dead. Developers should delete code that is definitely not being used. If you need it later, you can retrieve the previous version from the source control archive. If you do not have a source control tool in place, this situation is an example of the need for it.

If you find a section of code that is a candidate for “dead code”, your responsibility is to validate if the business logic is or is not used by the business and keep or delete it accordingly.

- ▶ Copy-paste technique

When the code is hard to understand, programmers can feel tempted to look for something that looks similar to what they need, copy and paste it and modify a few lines. Copy-paste is useful for text editing, but in software development it is a bad practice that generates much duplicate code, including “dead code”.

- ▶ Insufficient testing

Old, monolithic applications can be hard to test. Given this situation, programmers usually do not design or use preferred testing practices. They focus on testing specific parts of the logic and not the whole application. A good set of tests can help you to identify dead code.

7.8.2 Dead code risks

Because dead code is code that is not being used, you might think that is harmless, but the truth is that it has many risks:

- ▶ Dead code makes the application harder to understand.

More code means that understanding the code is more difficult. Keeping dead code means wasting time and effort understanding code that is not being used. Most often, this code is complex.

- ▶ Bugs are hidden.

Keeping dead code means that you are carrying untested code with many hidden bugs. If this code is ever reactivated, this can be a problem. It can also cause problems if developers are copying and pasting this code into other source.

7.8.3 Dead code detection

The only way to identify dead code is through dynamic analysis, which means you must analyze the running of the program and identify which lines are ran and which lines are not. Currently, there are not many commercial tools that help identify which code lines are run. This section introduces the iTrace tool, which is an open source tool that is available for download from the following website:

<https://code.google.com/p/ibmprogramtracer/downloads/list>

7.8.4 The iTrace tool

iTrace is a dynamic analysis tool that tracks every executed line of a program by using the debugger APIs that are included in IBM i. A basic iTrace session uses the following process:

1. Start the iTrace session.
2. Run the application to analyze it.
3. End the iTrace session.
4. Recover the generated data.
5. Calculate coverage.

With the data that is recovered from this process, you can start analyzing which sections of code are not being run. Every section of possible dead code must be validated with the corresponding business unit.

To learn how to use iTrace, consider the program that is shown in Example 7-10.

Example 7-10 Code sample for tracing

```
p main_test      b
d numVar        s          10i 0
/free
numVar = 0;
```

```

numVar = 100;
numVar *= 80;

if numVar > 100;
  dsply ('Number is bigger than 100');
else;
  dsply ('Number is smaller than 100');
endif;
/end-free
p                e

```

To trace the code in Example 7-10 on page 216, complete these steps (in the same interactive session):

1. Begin the tracing session by running the following command:
ITRACE SET(*ON) PROGRAM(TRACETEST)
2. Run the program to trace by running the following command:
CALL TRACETEST
3. Stop the tracing session by running the following command:
ITRACE SET(*OFF)

After you complete these steps, you can retrieve the trace data from the output file. The output file can be defined in step 1 in the parameter **OUTFILE**. For this example, the output is generated in the table QTEMP/TRACEDATA. Figure 7-1 shows the result of the generated data.

	REGTIME	CODSTMT	CODESEQ	CODEDTA	PGMNAME	PGMTYPE	PROCNAME
1	2013-08-19 15:41:...	108	000108	p main_test b	TRACETEST	*PGM	MAIN_TEST ...
2	2013-08-19 15:41:...	113	000113	numVar = 0;	TRACETEST	*PGM	MAIN_TEST ...
3	2013-08-19 15:41:...	114	000114	numVar = 100;	TRACETEST	*PGM	MAIN_TEST ...
4	2013-08-19 15:41:...	115	000115	numVar *= 80;	TRACETEST	*PGM	MAIN_TEST ...
5	2013-08-19 15:41:...	117	000117	if numVar > 100;	TRACETEST	*PGM	MAIN_TEST ...
6	2013-08-19 15:41:...	118	000118	dsply ('Number is bigger than 100'); ...	TRACETEST	*PGM	MAIN_TEST ...
7	2013-08-19 15:41:...	119	000119	else;	TRACETEST	*PGM	MAIN_TEST ...
8	2013-08-19 15:41:...	121	000121	endif;	TRACETEST	*PGM	MAIN_TEST ...
9	2013-08-19 15:41:...	123	000123	p e	TRACETEST	*PGM	MAIN_TEST ...

Figure 7-1 iTrace generated data

With the generated data, you can run the command **CodCoverage** (included with iTrace) to obtain a coverage report, indicating how many times each line was run. Figure 7-2 shows this behavior.

	PGMNAME	MDLNAME	PROCNAME	CODSTMT	CODESEQ	CODEDTA	FREQEXEC
1	TRACETEST	TRACETEST	MAIN_TEST ...	108	000108	p main_test b	4
2	TRACETEST	TRACETEST	MAIN_TEST ...	113	000113	numVar = 0;	4
3	TRACETEST	TRACETEST	MAIN_TEST ...	114	000114	numVar = 100;	4
4	TRACETEST	TRACETEST	MAIN_TEST ...	115	000115	numVar *= 80;	4
5	TRACETEST	TRACETEST	MAIN_TEST ...	117	000117	if numVar > 100;	4
6	TRACETEST	TRACETEST	MAIN_TEST ...	118	000118	dsply ('Number is bigger than 100'); ...	4
7	TRACETEST	TRACETEST	MAIN_TEST ...	119	000119	else;	4
8	TRACETEST	TRACETEST	MAIN_TEST ...	120	000120	dsply ('Number is smaller than 100');...	0
9	TRACETEST	TRACETEST	MAIN_TEST ...	121	000121	endif;	4
10	TRACETEST	TRACETEST	MAIN_TEST ...	123	000123	p e	4

Figure 7-2 iTrace code coverage report

7.8.5 Using caution when removing code

It is important to recognize that this tool reports only lines of code that were run during the test when the tracing tool was in use. It is critical to the success of using this tool to use a good and complete set of test cases to be more certain about which parts of the code are candidates for dead code. If your test scenarios are not designed well and validated with the corresponding business users, code might be identified as “dead” that was overlooked in the test scenarios. In this case, removing the suspected dead code can break the application.

For more information about iTrace, see the following website:

<https://code.google.com/p/ibmprogramtracer/>

7.9 Cyclomatic complexity and other metrics

Introduced in 1976 by Thomas J. McCabe, *cyclomatic complexity* is one of the most common metrics to objectively measure how complex a program is. Cyclomatic complexity measures the number of paths through a program. It counts how many control flow changes are in your code.

Why is this metric important? The level of complexity determines how easy it is to test, understand, and maintain a section of code. So, to write modern code, you must be aware of how complex your modern application is. It can also help you measure a refactoring project through a more engineered approach.

How can you calculate it? There are some commercial tools such as Databorough X-Analysis that can help you calculate the cyclomatic complexity for any source file. However, even if you do not have a tool, this should not be an excuse to ignore it. In a modernization project, it is essential to measure constantly. So, if a commercial tool is unavailable, you can code it yourself.

According to the Software Engineering Institute, program complexity should be kept under 50 points. Otherwise, your code represents a challenge to test and maintain. Use this number as a reference point.

7.9.1 Other relevant metrics

In addition to cyclomatic complexity, modern code can be measured through the usage of other metrics. In the software engineering product quality standard ISO/IEC 9126-3, you can find many metrics that can guide you in any modernization project:

- ▶ Modularity

Used to measure the number of modules that are functionally related and how coupled the application is.

- ▶ Program size

Used to measure the program scale, which based on the number of operators and operands that are present in the code.

- ▶ Average module size
Used to measure the average size of the modules in the application. It should consider only the executable statements of the application.
- ▶ Program statements
Measures how large the application is in terms of executable lines of code.

Although using standard metrics can help you, it is important that you consider defining custom metrics for the context of the language and platform. For example, if you are working on RPG, you can define a set of ad hoc metrics:

- ▶ Fixed form statements
Counts the number of statements that are coded in fixed form. This metric should be kept as low as possible if you want to build more maintainable code.
- ▶ Free format statements
Measures the number of statements in free format. There is no excuse for not coding in free format.
- ▶ Comments
Measuring how many comments are in the source code might provide interesting data, for example, a program with an excess of comments might correlate with high complexity or with obsolete code.
- ▶ Obsolete opcodes
Opcodes that are not supported in free form can be considered obsolete. Measuring how many obsolete opcodes are present in a specific module can help you estimate how hard it is to modernize it or how effectively an automated conversion tool can migrate code from fixed form to free form.
- ▶ Modern operators
Complementary to the obsolete opcodes metric. Measuring how many modern operators (that is, how many operation codes are valid for use in free form logic) are in your code can help you determine how easy it is to migrate to free format. Combining both metrics can give you a more accurate estimation.
- ▶ In-line declarations
In-line variable declarations, whether declared using the result field length or by using ***LIKE DEFINE**, make code less readable and far more difficult to maintain. Moving all data declarations to a single declaration location before the logic begins is relatively easy to do and brings many benefits over the lifetime of the code.
- ▶ Indicator usage
There is a general agreement that using traditional numbered indicators should be avoided as much as possible. Similar to the obsolete opcodes metric, the indicator usage can give you a sense of how difficult it is to convert your code. These metrics should count conditional, resulting, and numbered indicators.

To automatically calculate these ad hoc metrics, there is an open source tool that is hosted at <https://code.google.com/p/rpg-code-metrics/> where you can find more information and usage instructions.

7.9.2 Putting it all together

Metrics are valuable when they are used together. When you have all these metrics in place, with the small help of a spreadsheet, you can build a modernization dashboard that can become an essential tool for the project, especially for any progress update that the managers need to see. Figure 7-3 shows a simple example of a modernization dashboard.

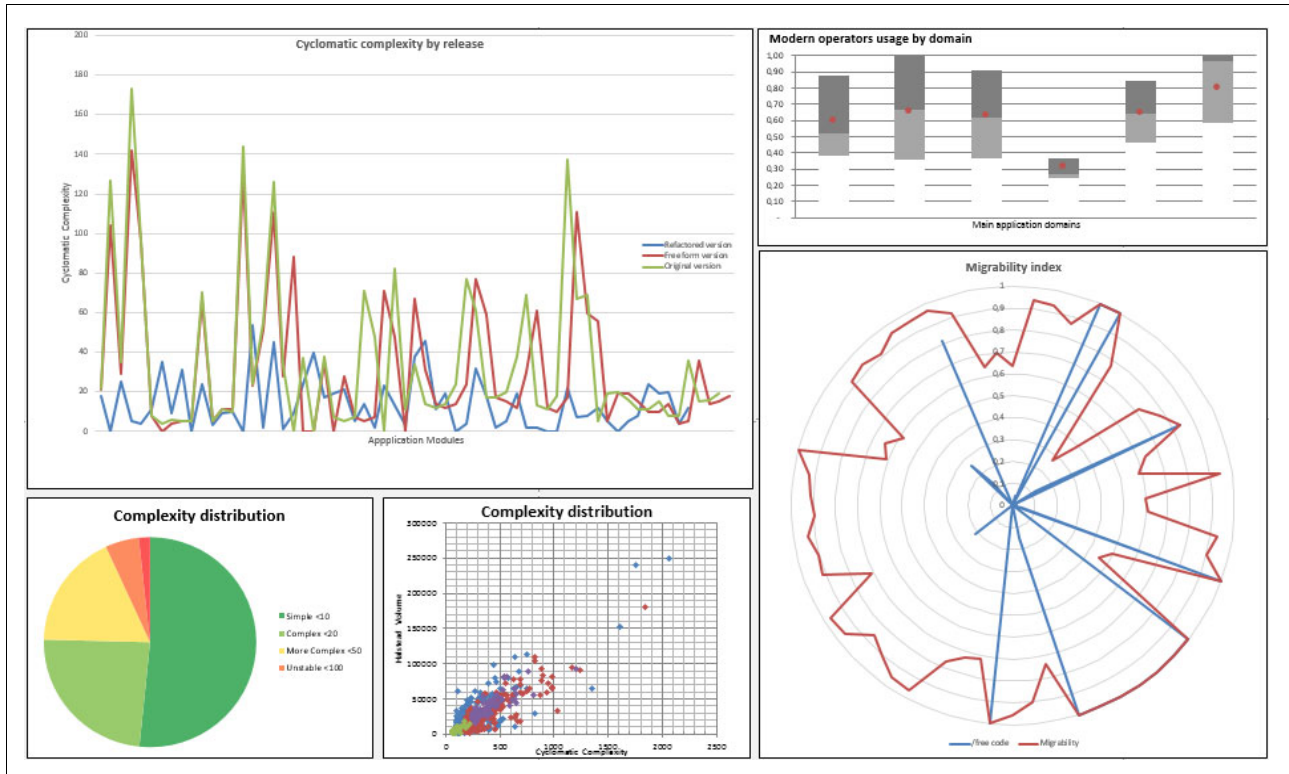


Figure 7-3 Modernization dashboard

Metrics can also be used to identify software development tendencies or coding skills gaps for the developers in your company. With this data, training efforts can be more precisely planned and evaluated afterward.



Data-centric development

Modernization does not pertain only to updating the user interface and application code.

Starting the modernization process by updating the database is another good approach. This action means that you are focusing on the data, and using the database where possible to act on the data. This process keeps the application focused on the business logic and the user interface. Any database modernization project can be a baseline only after a baseline is established. It is impossible to plan a journey without knowing the starting point. There are a range of starting points. For example, some companies might use a design that was created when the applications were first created years ago and was never updated. Other companies might find their starting point to be in a full relational design, to fourth or fifth normal form. However, for most companies, the starting point falls somewhere in the spectrum between these two points.

The original database designs might have come from an S/36 environment. This origin implies that the files are programs that are based on a flat file design. If the design is from the S/38 or early days of the AS/400, chances are that the database design was created one time and has lost any resemblance to that original design over time. Programmers can be good at adding a function or extending something after they do only a cursory review of the effect to the overall design. Although these designs continue to work on IBM i, neither approach takes full advantage of the power of DB2 for i.

Since the announcement of the AS/400, 25 years ago, IBM has continued to add new features and new functions to the database with each new release and technology refresh. A contemporary design is critical to take advantage of these new functions and to experience the performance improvements inherent in the updates. It is time to look at a *data-centric* view of development.

The name *sealinc*, which is used in this publication, was chosen as a fictitious name. It is used for instructional purposes only.

8.1 Set-based thinking

Traditionally, data retrieval in programs is achieved by using record level access (READ, WRITE, UPDATE, and so on). This section introduces ways to step beyond record level access (RLA) and use the concept of *data sets*. We look at set based concepts, function that is built into the database, and tools that you can use.

There have been many enhancements added to the database over the years that can provide great benefits:

- ▶ Referential integrity
- ▶ Indexes
- ▶ Views
- ▶ Commitment control
- ▶ Auto-generated columns

One of the most important improvements to the database over the years is the advancement of SQL. When we talk about a modern database, SQL is a requirement. This is not to suggest that native database access should be forbidden, but instead that it should use the correct tool for the job.

Traditional record access for small data sets can be effective. But, as data sets become larger, the effectiveness of native access can diminish. Additionally, your applications are required to take more responsibility for processing data across multiple tables. This processing can lead to complicated application code that can cause performance issues.

This situation is where SQL must be used. The beauty of SQL is that it uses the system or operating system versus the application. Many complicated data access routines can be replaced by SQL, which allows the system to figure out the indexes that make the most sense to retrieve the wanted data. The SQL engine on IBM i has undergone significant development focus over the past few years. The database has become better at creating and maintaining indexes to optimize your data access. The more records that you need to process, and views you need to combine, the better SQL can perform. In addition to the optimized indexing, SQL can use the multi-threading capabilities of the system without causing RPG and COBOL program (which are single-threaded) issues.

8.2 Defining physical containers for sets (SQL DDL)

This section describes the containers that are used when using a set-based process. It is important to understand DDLs and how they fit into this description.

8.2.1 Referential integrity

Referential integrity is an important aspect to the database modernization process. It is also the basis for the data-centric programming approach. Referential integrity is the vehicle for integrating the business logic into the database.

Referential integrity deals with relationships between data values in a relational database. These data relationships are closely tied with business rules. For example, a set of tables shares information about a corporation's customers and invoices. Occasionally, a customer's ID changes. You can define a referential constraint to state that the ID of the customer in a table must match a customer ID in the invoice information table. This constraint prevents insert, update, or delete operations that might otherwise result in missing customer information, as shown in Figure 8-1.

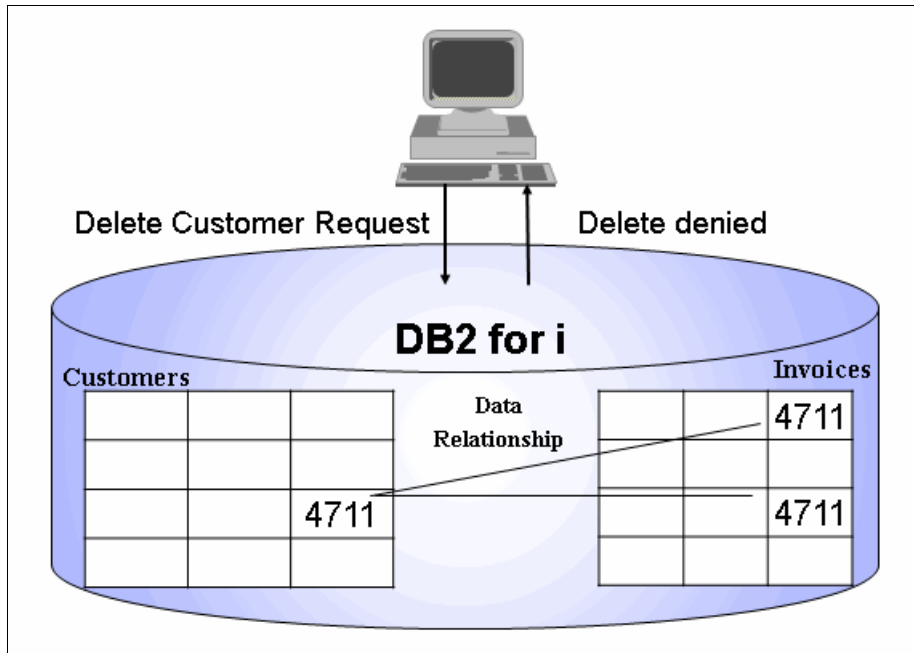


Figure 8-1 Referential integrity

Referential integrity is based on constraints. These constraints are basically rules that are created and managed by the database manager. These constraints can help you protect the integrity of your data.

Types of constraints

There are three types of constraints:

- ▶ Primary and unique key constraints

These constraints are used to prevent duplicate entries on a column from being entered into the database. The main difference between a primary key and unique is that a primary key does not allow NULL values in the data.

- ▶ Referential integrity constraints

Referential integrity defines the relationships between tables and ensures that these relationships remain valid. For example, a referential integrity constraint prevents an order from being inserted for an order that does not reference a valid customer number.

- ▶ Check constraints

You can check the constraints to ensure that column data does not violate the rules that are defined for the column or they can be used to allow only a certain set of allowed values into a column. For example, you can enforce that the order quantity amount is always greater than zero and less than 1000, or you can ensure that the state is valid.

Concepts of referential integrity

Here are the concepts for referential integrity:

- ▶ A parent key is a field or a set of columns in a table that must be unique, ascending, and cannot contain null values. Both a primary and unique key can be the parent key in a referential constraint. Only the primary key constraint enforces the “must be NULL” rule.
- ▶ A foreign key constraint is also known as a referential integrity constraint. It is a column or a set of columns in a table whose value, if not null, must match a value of the parent key in the related referential constraint. The value of a foreign key is null if at least one of the key fields is null.
- ▶ When defining a referential integrity constraint you must explicitly designate either a RESTRICT rule (a delete of the parent row is not allowed if a dependent row exists) or CASCADE (the delete of the parent is allowed and all dependent rows are also deleted). The referential integrity constraint can be disabled or enabled through the Change PF Constraint (CHGPF CST) command.

Figure 8-2 shows referential integrity concepts.

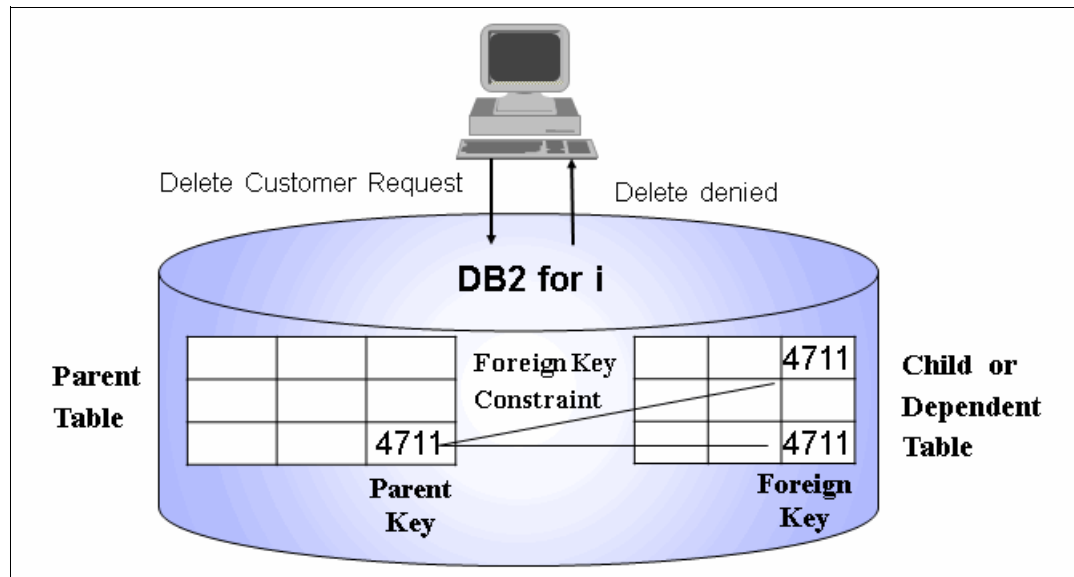


Figure 8-2 Referential integrity concepts

For a more detailed description about how to implement referential integrity in DB2 for i, see *Advanced Functions and Administration on DB2 Universal Database for iSeries*, SG24-4249.

Examples

This section shows some examples about how to implement these concepts in your applications.

When you are create a table, you can create a unique constraint in your SQL statement, as shown in Example 8-1.

Example 8-1 Creating a unique constraint

```
CREATE TABLE agents (  
  agent_no INTEGER DEFAULT NULL ,  
  agent_name VARCHAR(64) DEFAULT NULL ,  
  agent_pwd VARCHAR(64) DEFAULT NULL,
```

```
CONSTRAINT customer_key  
UNIQUE (agent_no))
```

Alternatively, you can create a primary constraint, as shown in Example 8-2.

Example 8-2 Creating a primary key constraint

```
CREATE TABLE customers (  
customer_no FOR COLUMN cust_no INTEGER DEFAULT NULL ,  
customer_name FOR COLUMN cust_name VARCHAR(64) CCSID 37 DEFAULT NULL , address  
VARCHAR(150) CCSID 37 NOT NULL DEFAULT '' ,  
city CHAR(50) CCSID 37 DEFAULT NULL ,  
state CHAR(2) CCSID 37 DEFAULT NULL ,  
zipcode CHAR(9) CCSID 37 DEFAULT NULL ,  
telephone CHAR(20) CCSID 37 DEFAULT NULL ,  
credit_card FOR COLUMN cred_card CHAR(30) CCSID 37 DEFAULT NULL ,  
cc_number CHAR(20) CCSID 37 DEFAULT NULL ,  
exp_date CHAR(20) CCSID 37 DEFAULT NULL ,  
pref_airline_id FOR COLUMN pref_airln CHAR(10) CCSID 37 DEFAULT NULL ,  
ff_number CHAR(20) CCSID 37 DEFAULT NULL, CONSTRAINT customer_no  
PRIMARY KEY (customer_no))
```

With SQL, the equivalent action takes place with the following two **ALTER TABLE** statements:

```
ALTER TABLE customers  
ADD CONSTRAINT customers_unq_customer_number  
UNIQUE KEY( customer_no );
```

```
ALTER TABLE agents  
ADD CONSTRAINT agents_pk_agent_number  
PRIMARY KEY( agent_no );
```

The “do no harm” approach

Constraints are a powerful tool and can be applied to existing models, but you must be careful; before you begin using constraints to enforce business rules, you must consider the impact on existing applications because the same business rules that constraints enforce often in application code. For example, a referential constraint might require that all orders have valid customer numbers. To enforce this business rule using RPG, before you create an order, use a **SETLL** operation against a file containing the Customer unique key to verify that the customer number exists. If it does not, you display an error message and the order is not added.

To understand how constraints work with other application design options, you might find it helpful to consider how you apply duplicate key rules. It is a common practice in IBM i shops to define duplicate key rules when a physical file is created. These rules ensure that no matter what application modifies the file, or what errors or malicious code that application contains, the database is never corrupted with duplicate keys. This requirement does not mean that your applications cease to check for duplicate keys, but it does mean that the database is protected even if your applications are bypassed or are incorrect. This is the reason for the duplicate key rule, which is a primary key or unique constraint in SQL terms. It ensures that the file contains valid data regardless of how it is updated.

If you apply the logic you use for imposing duplicate key rules to referential and check constraints, it is much easier to see how you can begin using these database features. Begin by defining referential and check constraints for business rules that are well-defined and consistently enforced by your applications. A constraint is always enforced, so you want to avoid imposing constraints if the existing data contains violations, or if the applications do not conform to the constraint rules.

When you have defined a constraint, the next question is how to deal with constraint violations in your applications. If you have an application that checks a business rule that is also enforced by a constraint, it is often preferable to leave that application code intact and accept the slight performance penalty that is incurred by checking twice: once in the constraint and once in the application code.

If you are writing applications, consider checking for constraint violations instead of using data validation techniques, such as chaining to a master file. However, you might find that there are situations where the existing methods of checking for errors make more sense than checking for constraint violations. For example, if you are already chaining to the customer master file to retrieve the customer name, it makes sense to handle invalid customer numbers at the same time, using the same methodology that you currently use.

Another issue to consider is how easily you can determine which constraint failed. SQL, RPG, and COBOL all signal constraint violations. However, if multiple constraints are assigned to a table, as is generally the case, you must retrieve the information about which constraint failed by using the Receive Program Message API. In addition, constraint violations are reported when the first violation is encountered. Therefore, if you must validate an entire panel of information and report all errors to the user, checking for constraint violations in your application can be difficult.

Finally, although the duplicate key comparison works well for most constraints, some referential constraints do more than simply prevent invalid data from being stored in a table. If you define a constraint that cascades (for example, deleting all order line rows when the corresponding Order Header row is deleted), you most likely want to remove any application code that performs the same function as the constraint.

Even if you decide never to check for constraint violations in your RPG or COBOL applications, you might still want to impose constraints. Doing so makes your business rules accessible to applications that are running on other platforms or written in languages such as Java. It protects your data from corruption and it improves application portability because constraints are a standard database capability.

8.2.2 Indexes

An *index* is a set of pointers to rows within a database table. Each index is based on the values of data in one or more table columns. An index is an object that is separate from the data in the table. When an index is created, the database manager builds this structure and maintains it automatically.

An index has a name and might have a different system name. The system name is the name that is used by the IBM i operating system. Either name is acceptable wherever an index-name is specified in SQL statements.

The database manager uses two types of indexes: binary radix tree indexes and EVIs.

Binary radix tree index

Binary radix tree indexes provide a specific order to the rows in a table, as shown in Figure 8-3. The database manager uses them to perform the following tasks:

- ▶ Improve performance: In most cases, access to data is faster than without an index.
- ▶ Ensure uniqueness: A table with a unique index cannot have rows with identical keys.

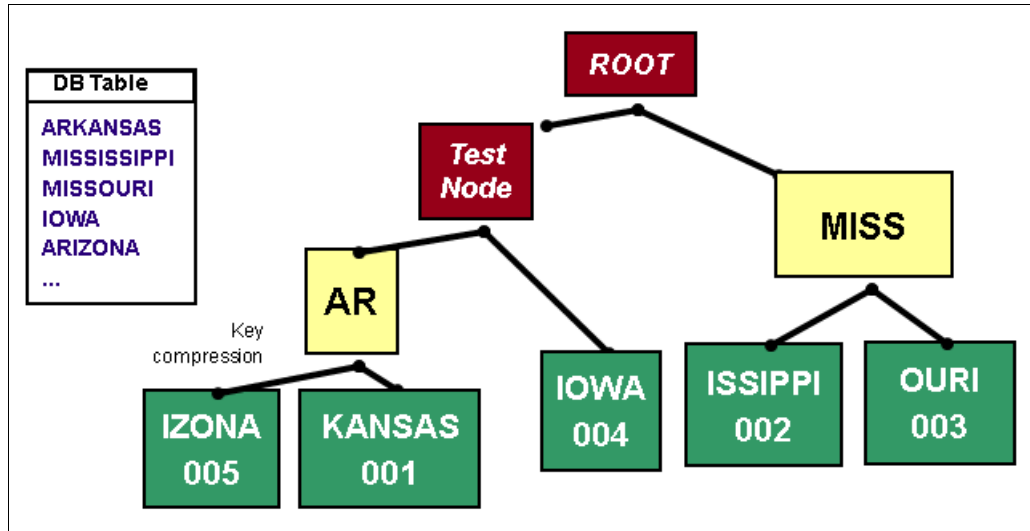


Figure 8-3 Binary radix tree indexes

Encoded vector index

Encoded vector indexes (EVIs) do not provide a specific order to the rows of a table. The database manager uses these indexes only to improve performance.

An EVI is used to provide fast data access in decision support and query reporting environments.

EVIs are a complementary alternative to existing index objects (binary radix tree structure, logical file, or SQL index) and are a variation on bitmap indexing. Because of their compact size and relative simplicity, EVIs provide for faster scans of a table that can also be processed in parallel.

An EVI is a data structure that is stored as two components, as shown in Figure 8-4:

- ▶ The symbol table contains statistical and descriptive information about each distinct key value that is represented in the table. Each distinct key is assigned a unique code, either 1 byte, 2 bytes, or 4 bytes in size.
- ▶ The vector is an array of codes that are listed in the same ordinal position as the rows in the table. The vector does not contain any pointers to the rows in the table.

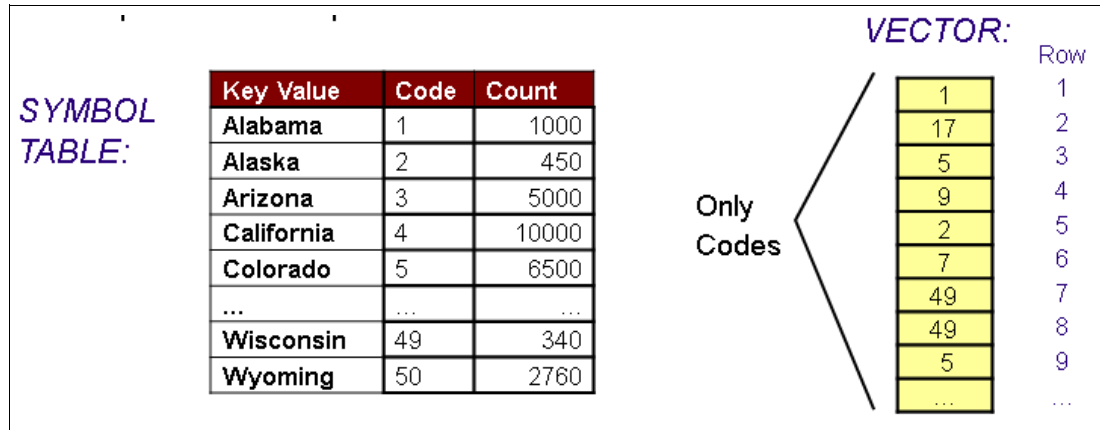


Figure 8-4 EVI structure

Advantages of EVIs

Here are some of the reasons to consider using EVIs:

- ▶ Require less storage.
- ▶ Can have better build times than radix tree indexes, especially if the number of unique values in the columns that are defined for the key is relatively small.
- ▶ Provide more accurate statistics to the query optimizer.
- ▶ Considerably better performance for certain grouping types of queries.
- ▶ Good performance characteristics for decision support environments.
- ▶ Can be further extended for certain types of grouping queries with the addition of **INCLUDE** values. Provides ready-made numeric aggregate values that are maintained in real time as part of index maintenance. **INCLUDE** values become an extension of the EVI symbol table. Multiple **INCLUDE** values can be specified over different aggregating columns and maintained in the same EVI if the group by values are the same. This technique can reduce overall maintenance.

Disadvantages of EVIs

There are a few situations where it is best not to use an EVI. Consider the following limitations:

- ▶ Cannot be used in ordering.
- ▶ Usage for grouping is specialized. The EVI indexes support the following items:
 - COUNT, DISTINCT requests over key columns
 - Aggregate requests over key columns where all other selections can be applied to the EVI symbol table keys
 - **INCLUDE** aggregates
 - MIN or MAX, if the aggregating value is part of the symbol table key
- ▶ Use with joins is always done in cooperation with hash table processing.

When to create EVIs

There are several situations in which to consider creating EVIs. Consider creating EVIs when any one of the following conditions is true:

- ▶ You want to gather live statistics.
- ▶ A full table scan is selected for the query.
- ▶ Selectivity of the query is 20 - 70% and using skip sequential access with dynamic bitmaps speeds up the scan.
- ▶ When a star schema join is expected to be used for star schema join queries.
- ▶ When grouping or distinct queries are specified against a column, the columns have few distinct values and only the COUNT column function, if any, is used.
- ▶ When ready-made aggregate results are grouped by the specified key columns that benefit query performance.

When creating an EVI, it is preferable to create EVIs with the following items specified:

- ▶ Single key columns with few distinct values expected
- ▶ Keys columns with a low volatility (do not change often)
- ▶ Maximum number of distinct values that are expected from using the WITH *n* DISTINCT VALUES clause
- ▶ Single key over foreign key columns for a star schema model

Recommendations for EVI usage

EVIs are a powerful tool for providing fast data access in decision support and query reporting environments. To ensure the effective use of EVIs, follow these guidelines:

- ▶ Create EVIs on the following items:
 - Read-only tables or tables with a minimum of **INSERT**, **UPDATE**, and **DELETE** activity.
 - Key columns that are used in the WHERE clause.
 - Single key columns that have a relatively small set of distinct values.
 - Multiple key columns that result in a relatively small set of distinct values.
 - Key columns that have a static or relatively static set of distinct values.
 - Non-unique key columns, with many duplicates.
- ▶ Create EVIs with the maximum byte code size expected:
 - Use the “WITH *n* DISTINCT VALUES” clause on the **CREATE ENCODED VECTOR INDEX** statement.
 - If unsure, use a number greater than 65,535 to create a 4-byte code. This method avoids the EVI maintenance that is involved in switching byte code sizes.
 - EVIs with **INCLUDE** always create with a 4-byte code.
- ▶ When loading data:
 - Drop EVIs, load data, and create EVIs.
 - The EVI byte code size is assigned automatically based on the number of distinct key values that are in the table.
 - The symbol table contains all key values, in order, with no keys in the overflow area.
 - EVIs with **INCLUDE** always use 4-byte code.

Adding **INCLUDE** values to existing EVIs

An EVI index with **INCLUDE** values can be used to supply ready-made aggregate results, as shown in Figure 8-5. The existing symbol table and vector are still used for table selection, when appropriate. They are used for skipping sequential plans over large tables, or for index ANDing and ORing plans. If you have EVIs, consider creating ones with more **INCLUDE** values, and then drop the pre-existing index.

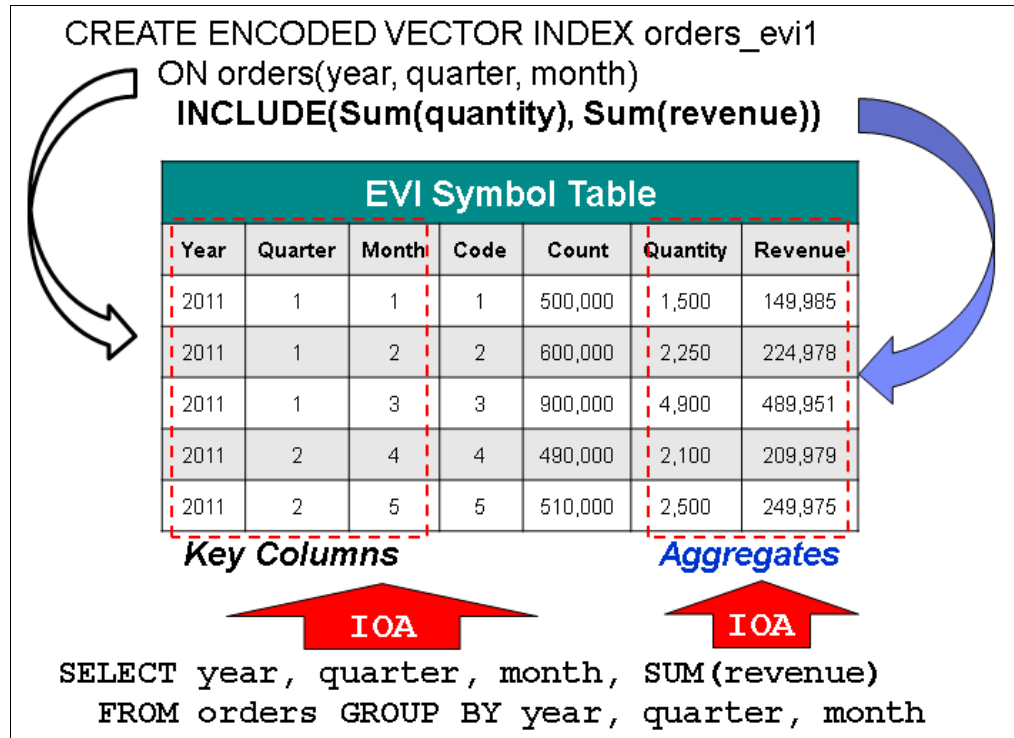


Figure 8-5 EVI Indexes with **INCLUDE** clause

In Figure 8-5, the query needs the information about the following columns: Year, Quarter, Month, and Revenue. Because the index has the same columns, the optimizer decided to solve the query by using only the index; this access method is called Index Only Access (IOA).

The access method IOA is important for saving disk I/O, processor, and memory.

Specifying multiple **INCLUDE** values

If you need different aggregates over different table values for the same **GROUP BY** columns that are specified as EVI keys, define those aggregates in the same EVI. This definition cuts down on maintenance costs and allows for a single symbol table and vector, as shown in Example 8-3.

Example 8-3 Creating multiple values for **INCLUDE**

```
Select SUM(revenue) from sales group by Country
Select SUM(costOfGoods) from sales group by Country, Region
Both queries could benefit from the following EVI:
CREATE ENCODED VECTOR INDEX eviCountryRegion on Sales(country,region)
INCLUDE(SUM(revenue), SUM(costOfGoods))
```

The optimizer does additional grouping (regrouping) if the EVI key values are wider than the corresponding GROUP BY request of the query. This additional grouping is the case in the first example query.

If an aggregate request is specified over null capable results, an implicit COUNT over that same result is included as part of the symbol table entry. The COUNT is used to facilitate index maintenance when a requested aggregate needs to reflect. It can also assist with pushing aggregation through a join if the optimizer determines that this push is possible. The COUNT is then used to help compensate for less join activity because of the pushed down grouping.

Considerations for indexes

You can create any number of indexes. However, because the indexes are maintained by the system, many indexes can adversely affect performance. One type of index, the EVI, allows for faster scans that can be more easily processed in parallel.

If an index is created that has the same attributes as an existing index, the new index shares the existing indexes' binary tree. Otherwise, another binary tree is created. If the attributes of the new index are the same as another index, except that the new index has fewer columns, another binary tree is still created. It is still created because the extra columns prevent the index from being used by cursors or UPDATE statements that update those extra columns.

Index advisor

The query optimizer analyzes the row selection in the query and determines, based on the default values, if creation of a permanent index improves performance. If the optimizer determines that a permanent index might be beneficial, it returns the key columns that are necessary to create the suggested index.

The optimizer can perform a radix index probe over any combination of the primary key columns, plus one additional secondary key column. Therefore, it is important that the first secondary key column is the most selective secondary key column. The optimizer uses radix index scan with any of the remaining secondary key columns. Although radix index scan is not as fast as radix index probe, it can still reduce the number of keys that are selected. It is recommended that secondary key columns that are fairly selective be included.

Determine the true selectivity of any secondary key columns and whether you include those key columns in the index. When you are building the index, make the primary key columns the leftmost key columns, followed by any of the secondary key columns that are chosen, prioritized by selectivity.

Note: After creating the suggested index and running the query again, it is possible that the query optimizer does choose to use the suggested index. When suggesting indexes, the CQE optimizer considers only the selection criteria. It does not include join, ordering, and grouping criteria. The SQE optimizer includes selection, join, ordering, and grouping criteria when suggesting indexes.

You can access index advisor information in several ways:

- ▶ The index advisor interface in System i Navigator
- ▶ SQL performance monitor Show statements
- ▶ Visual Explain interface
- ▶ Querying the database monitor view 3020 - Index advised

An example of the Index Advisor window is shown in Figure 8-6.

Table for Which Index was Advised	Schema	System Schema	System Name	Partition	Keys Advised	Leading Keys Order Independent	Advised Index Type	Last Advised for Query Use
QASZRAIRD	QUSRSYS	QUSRSYS	QASZRAIRD	For all partitions	COMPID, FULLPATH	COMPID, FULLPATH	Binary Radix	8/1/13 3:09:12 PM
DUMP	XMLSERVLOG	XMLSERVLOG	DUMP	For all partitions	KEY, LOG	KEY	Binary Radix	8/18/13 2:17:35 AM
QDBRGZ_QUSRSYS_QASX...	QRECOVERY	QRECOVERY	QDBRG38293	For all partitions	STEP	STEP	Binary Radix	8/20/13 10:00:20 PM
QA1AHS	QUSRBRM	QUSRBRM	QA1AHS	For all partitions	BKHSYS, BKPKN, BKJNBR, B...	BKHSYS, BKPKN, BKJNBR, B...	Binary Radix	5/11/13 2:52:24 PM
QA1AHS	QUSRBRM	QUSRBRM	QA1AHS	For all partitions	BKHSYS, BKMFNB, BKPKN, ...	BKHSYS, BKMFNB, BKPKN, ...	Binary Radix	5/11/13 2:52:24 PM
QDBRGZ_QUSRSYS_QASX...	QRECOVERY	QRECOVERY	QDBRG38293	For all partitions	STEP	STEP	Binary Radix	8/20/13 10:00:19 PM
QDBRGZ_QUSRSYS_QASX...	QRECOVERY	QRECOVERY	QDBRG38293	For all partitions	STEP	STEP	Binary Radix	8/20/13 10:00:19 PM
QDBRGZ_QUSRSYS_QASX...	QRECOVERY	QRECOVERY	QDBRG38293	For all partitions	STEP	STEP	Binary Radix	8/20/13 10:00:20 PM
QALWIRR	QLWIRADM	QLWIRADM	QALWIRR	For all partitions	REQUESTNAME	REQUESTNAME	Binary Radix	8/20/13 4:05:09 PM
QDBRGZ_QUSRSYS_QASX...	QRECOVERY	QRECOVERY	QDBRG38293	For all partitions	STEP	STEP	Binary Radix	8/20/13 10:00:19 PM
LDAP_ENTRY	QUSRDIRDB	QUSRDIRDB	LDAP_ENTRY	For all partitions	DN_TRUNC, DN	DN_TRUNC	Binary Radix	8/20/13 4:04:33 PM
QALWIUR	QLWIRADM	QLWIRADM	QALWIUR	For all partitions	USERNAME	USERNAME	Binary Radix	2/15/13 2:00:32 PM
QA1AHS	QUSRBRM	QUSRBRM	QA1AHS	For all partitions	BKHSYS, BKHVSQ, BKHSWT, ...	BKHSYS, BKHVSQ, BKHSWT, ...	Binary Radix	5/11/13 2:52:24 PM
QDBRGZ_QUSRSYS_QA1P...	QRECOVERY	QRECOVERY	QDBRG22639	For all partitions	STEP	STEP	Binary Radix	8/1/13 12:02:21 AM
ANIMAL	XMLSERVTST	XMLSERVTST	ANIMAL	For all partitions	WEIGHT	WEIGHT	Binary Radix	8/21/13 10:17:03 AM
LOG	XMLSERVLOG	XMLSERVLOG	LOG	For all partitions	KEY, LOG	KEY, LOG	Binary Radix	8/18/13 2:17:33 AM
ITDSRDBMHISTORY	QUSRDIRDB	QUSRDIRDB	ITDSR00001	For all partitions	FEATURE	FEATURE	Binary Radix	8/20/13 4:04:32 PM
ITDSRDBMHISTORY	QUSRDIRDB	QUSRDIRDB	ITDSR00001	For all partitions	RELEASE, FEATURE	RELEASE, FEATURE	Binary Radix	8/20/13 4:04:32 PM
LDAP_DESC	QUSRDIRDB	QUSRDIRDB	LDAP_DESC	For all partitions	AED, DEID	AED	Binary Radix	8/20/13 4:04:34 PM
QA1AHS	QUSRBRM	QUSRBRM	QA1AHS	For all partitions	BKHSYS, BKJNBR, BKPKN, B...	BKHSYS, BKJNBR, BKPKN, B...	Binary Radix	6/28/12 2:23:03 AM
QA1AHS	QUSRBRM	QUSRBRM	QA1AHS	For all partitions	BKHSYS, BKJNBR, BKMFNB, ...	BKHSYS, BKJNBR, BKMFNB, ...	Binary Radix	6/28/12 2:23:03 AM
QA1AHS	QUSRBRM	QUSRBRM	QA1AHS	For all partitions	BKHSYS, BKHVSQ, BKHSWT, ...	BKHSYS, BKHVSQ, BKHSWT, ...	Binary Radix	5/11/13 2:52:24 PM
ANIMAL	XMLSERVTST	XMLSERVTST	ANIMAL	For all partitions	ID	ID	Binary Radix	8/21/13 10:16:52 AM
ANIMALS	DB2	ANIMALS	DB2	For all partitions	WEIGHT	WEIGHT	Binary Radix	7/26/13 12:50:46 PM
QASZRAIRC	QUSRSYS	QUSRSYS	QASZRAIRC	For all partitions	VENDOR, VERSION, COMPN...	VENDOR, VERSION, COMPN...	Binary Radix	7/10/12 9:17:56 PM
QA1ALM	QUSRBRM	QUSRBRM	QA1ALM	For all partitions	QLQUAL, QLTYPE, QLNAME	QLQUAL, QLTYPE	Binary Radix	8/18/13 7:00:04 AM

Figure 8-6 Index Advisor window

8.2.3 Views

A view is a named specification of a result table. The specification is a **SELECT** statement that is effectively run whenever the view is referenced in an SQL statement. Thus, a view can be thought of as having columns and rows just like a database table. For retrieval, all views can be used just like base tables. Whether a view can be used in an **INSERT**, **UPDATE**, or **DELETE** operation depends on its definition.

An index cannot be created for a view. However, an index that is created for a table on which a view is based might improve the performance of operations on the view.

When the column of a view is directly derived from a column of a base table, that column inherits any constraints that apply to the column of the base table. For example, if a view includes a foreign key from its base table, **INSERT** and **UPDATE** operations using that view are subject to the same referential constraints as the base table. Likewise, if the base table of a view is a parent table, **DELETE** operations using that view are subject to the same rules as **DELETE** operations on the base table. A view also inherits any triggers that apply to its base table. For example, if the base table of a view has an update trigger, the trigger is fired when an update is performed on the view.

A view has a name and might have a different system name. The system name is the name that is used by the IBM i operating system. Either name is acceptable wherever a view name is specified in SQL statements.

A column of a view has a name and might have a different system column name.

The system column name is the name that is used by the IBM i operating system. Either name is acceptable wherever column-name is specified in SQL statements.

One approach is to use only views and allow none of the programs to access the physical data model (the tables) directly. This is the ideal approach for database administrators (DBAs) to consider implementing. In this type of implementation, DBAs must determine what columns are eligible to be queried from both users and applications. After that analysis is completed, the DBA can create views that project only those columns that users and applications need to see.

Here are the attributes that provide more flexibility in selecting and processing data:

- ▶ CASE expression.
- ▶ Date/time functions.
- ▶ Grouping.
- ▶ Join processing.
- ▶ Views can be opened by native programs as nonkeyed logical files.
- ▶ Views can be directly accessed through ODBC/JDBC.
- ▶ Enables database programmers to mask complexity of the database to users.
- ▶ Provides a way to restrict access to the data in the table.
- ▶ Views can be used to define an externally defined data structure in programs.
- ▶ View structure can be used as the parameter to pass to or from the I/O procedures.

With these advantages in mind, the new SQL I/O procedures use views (instead of tables) to access the data.

Remember:

- ▶ Views are referenced just like tables in SQL statements.
- ▶ Views can be joined to other views.
- ▶ Views can be used to externally describe data structures.

8.2.4 Auto-generated column support

Whenever possible, it is best to allow DB/2 for i to do the work for you. The Database Management System (DBMS) automatically populates and manages preexisting data types. This provides significant advantages to your applications because they do not have to manage this data. These data types are a must for any data-centric application architecture.

Identity column

An identity column is an auto-generated numeric column that the DBMS increments for you as each row is inserted into the table. An identity column is defined as either SMALLINT, INTEGER, or BIGINT, depending on the expected number of rows the table has. You can specify a starting number, increment value, and maximum value for the DBMS to use. To ensure that the value is always unique, either define a unique index on the identity column or define the identity column as a primary key.

One advantage to the Identity Columns approach is that the DBMS does the work for you. It tracks the next value instead of requiring your applications to track a sequence value that is stored somewhere. The DBMS handles multiple requests of rows that are inserted into the table and ensures that the Identity Column is unique when inserted into a table.

Restrictions

After it is created, you cannot alter the table description to include an identity column.

If rows are inserted into a table with explicit identity column values specified, the next internally generated value is not updated, and might conflict with existing values in the table. Duplicate values generate an error message if the uniqueness of the values in the identity column is being enforced by a primary-key or a unique index that is defined on the identity column.

A table can have only one identity column defined.

Note: To get the value of the identity column that the system assigned to the last row that your program inserted into a table, run the following command:

```
exec sql values identity_val_local() into :working-field-1
```

Note: To get the next value that the system will assign if a row is inserted into a table, use the following code (where *table_schema* = *library_a* and *table_name* = *table_a*):

```
select next_identity_value  
from qsys2.syspstat
```

Usage

One of the primary uses of identity columns is to set up a Primary Key (PK)/Foreign Key (FK) relationship between two tables. The identity column is defined as a PK in *table_a* and *table_b* has a column of the same type (SMALLINT, INTEGER, and BIGINT) containing the value in *table_a*.

It is helpful to use identity columns to build relationships between tables instead of using business data because business rules and requirements change over time. If you are using business data as relationships between tables, you might run into a situation where the PK/FK relationship restricts growth to business needs.

Row change time stamp

A row change time stamp column is a system generated time stamp value that is automatically set when a row is inserted or changed in a table. The column must be defined as `TIMESTAMP` and must have the following text after the data type:

```
FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP NOT NULL
```

Every time a row is added or changed in a table with a row change time stamp column, the row change time stamp column value is set to the time stamp corresponding to the time of the insert or update operation.

Note: You do not need to know the column name to use the row change time stamp column in a SQL statement. Your SQL statement can look like this (where `ROW CHANGE TIMESTAMP FOR table_a > CURRENT TIMESTAMP - 24 HOURS`):

```
Select column_1, column_2, ROW CHANGE TIMESTAMP FOR table_a  
from table_a
```

This SQL statement lists every row that was inserted or changed in the last 24 hours.

Restrictions

A table can have only one row change time stamp defined.

Usage

Row change time stamp columns have many valuable uses. They include, but are not limited to, the following usage:

- ▶ Optimistic locking in commitment control scenarios.
 - Read a row without locking and allowing the user to update information in the maintenance screen.
 - When the user presses <SAVE>, you can read the row again and compare the ROW CHANGE TIMESTAMP column with the previous read record.
 - If the ROW CHANGE TIMESTAMP did not change, update the row.
 - If the ROW CHANGE TIMESTAMP changed, notify the user that the record has already been updated.
- ▶ Data replication scenarios where you need to know what rows have been changed since the last time the target database was updated.
- ▶ Data warehousing applications where you need to apply new or changed rows to a data warehouse.

Sequence object

The sequence object allows automatic generation of values. Unlike an identity column attribute, which is bound to a specific table, a sequence object is a global and stand-alone object that can be used by any tables in the same database.

Here is an example of creating a sequence object that is named ORDER_SEQ:

```
CREATE SEQUENCE order_seq START WITH 10 INCREMENT BY 10
```

When inserting a row, the sequence number must be determined through NEXT VALUE FOR SEQUENCE. For example, here we insert a row to the ORDERS table by using a value from the sequence object:

```
INSERT INTO orders(ordnum,custnum) VALUES( NEXT VALUE FOR order_seq, 123 )
```

Because the sequence is an independent object and is not directly tied to a particular table or column, it can be used with multiple tables and columns. Because of its independence from tables, a sequence can be easily changed through the SQL statement **ALTER SEQUENCE**.

The **ALTER SEQUENCE** statement generates or updates only the sequence object, and it does not change any data.

Rowid data type

A rowid is a value that uniquely identifies a row in a table. A column or a host variable can have a rowid data type. A rowid column enables queries to be written that navigate directly to a row in the table. Each value in a rowid column must be unique. The database manager maintains the values permanently, even across table reorganizations. When a row is inserted into the table, the database manager generates a value for the rowid column unless one is supplied. If a value is supplied, it must be a valid rowid value that was previously generated by either DB2 UDB for IBM OS/390® and IBM z/OS® or DB2 UDB for iSeries.

The internal representation of a rowid value is not apparent to the user. The value is never subject to CCSID conversion because it is considered to contain BIT data. Rowid columns contain values of the rowid data type, which returns a 40-byte VARCHAR value that is not regularly ascending or descending.

A table can have only one rowid. A row ID value can be assigned only to a column, parameter, or host variable with a rowid data type. For the value of the rowid column, the column must be defined as GENERATED BY DEFAULT, or OVERRIDING SYSTEM VALUE must be specified. A unique constraint is implicitly added to every table that has a rowid column that ensures that every rowid value is unique. A rowid operand cannot be directly compared to any data type. To compare the bit representation of a rowid, first cast the rowid to a character string.

Note: In RPG, there is no data type that directly matches with the rowid data type, but by using the keyword **SQLTYPE** in the Definition specifications, host variables can be defined to hold the rowid.

This example shows the definition of a host variable with the SQL data type rowid:

```
D MyRowId S SQLTYPE(ROWID)
```

Accessing auto-generated values by using INSERT on FROM

You can retrieve the values for rows that are being inserted, including DB2 for i auto-generated values, by specifying the **INSERT** statement in the FROM clause of a **SELECT** statement, as shown here:

```
SELECT identity_column, row_change_timestamp, sequence, row_id, ...
FROM FINAL TABLE (
    INSERT INTO T1 (required columns)
    VALUES(required values))
```

In addition to auto-generated values, you can include any of the following values:

- ▶ Any default values that are used for columns
- ▶ All values for all rows that are inserted by a multiple-row insert operation
- ▶ Values that were changed by a before insert trigger

8.3 Accessing and operating on sets (SQL DML)

Set-based data access has existed for over 20 years and is central to most object oriented (OO) and SQL-based languages. However, even the most modern RPG shops are slow to adopt these concepts and still rely on RLA and traditional I/O to process data. RLA does not scale as well and it is not as flexible as accessing data in sets through SQL statements.

This section looks at two different approaches to embedding SQL statements in High Level Language (HLL) programs to operate on sets. The two approaches are running *static* and *dynamic* SQL statements from within your program.

8.3.1 Static

Because SQL-described tables are physical objects and views are logical file objects, you can use these tables with traditional I/O; however, there are some restrictions (an SQL view cannot be ordered). The preferred method for accessing SQL defined objects is through SQL Data Manipulation Language (DML) statements that are embedded in your HLL source code.

In static SQL, the statement is partially processed at compile time. You can integrate host variables into the SQL statement, which are subsequently set at run time. The syntax is checked by the precompiler, and then the SQL statements are replaced by SQL function calls.

Static SQL statements are used in cases where the SQL statement is known ahead of processing time and does not need to change each time it is called or when an SQL statement (such as **SELECT INTO**) cannot be dynamically prepared.

For example, consider an SQL view that contains a JOIN and a GROUP BY clause to summarize specific columns of data from two or more tables. The format and order of the result set are always the same; however, the data is filtered by the usage of a WHERE clause. If you need a view to bring customer and order information together, and your program requires only specific columns, your **CREATE VIEW** statement might look like what is shown here:

```
CREATE VIEW CUSTORDRS AS
(SELECT FLIGHT_NO,CUSTOMER_NAME,CLASS, AGENT_NO,
  FROM CUSTOMERS
  INNER JOIN ORDERS
  ON ( CUSTOMERS.CUSTOMER_NO = ORDERS.CUSTOMER_NO )
GROUP BY FLIGHT_NO,CLASS, AGENT_NO
);
```

The embedded SQL in the HLL program can read from the CUSTORDRS view as follows:

```
SELECT * FROM CUSTORDRS
WHERE FLIGHT_NO = :vFlightNumber
ORDER BY CUSTOMER_NAME;
```

Static SQL is commonly used for the following tasks:

- ▶ To return one single row from a select statement into host variables
- ▶ To insert, update, or delete several rows by using one single SQL statement
- ▶ For other actions such as:
 - To declare global temporary tables
 - To create and drop temporary aliases
 - To grant temporary privileges
 - To set path or set schema

Static SQL returning a single row

If the result of an SQL statement is only one row, it can be directly returned into host variables in one of the following manners:

- ▶ **SELECT ... INTO**
- ▶ **SET HostVariable = SELECT ...**

SELECT ... INTO

The **SELECT INTO** statement produces a result table that consists at most of one row, and assigns the values in that row to host variables with the following conditions:

- ▶ If a single row is returned, **SQLCODE** is set to 0 or a value 1 - 99, and the appropriate values are assigned to the host variables.
- ▶ If the table is empty, the statement assigns +100 to **SQLCODE** and '02000' to **SQLSTATE**. If you do not use indicator variables to detect NULL values, the host variables are not updated; otherwise, NULL is returned.
- ▶ If the result consists of more than one row, **SQLCODE -811** is returned, but the host-variables are updated with the results from the first row.

Example 8-4 shows how the total count of an order is calculated and returned within a **SELECT ... INTO** statement.

Example 8-4 Results from a SELECT...INTO statement

```
Dcl-s Agent Packed(9);
Dcl-s TotalFlights Packed(9);

Agent = 1234;
Exec SQL
    SELECT COUNT(*)
    INTO :TotalFlights
    FROM ORDERS
    WHERE AGENT_NO = :Agent;
```

SET: HostVariable = (SELECT ...)

It is possible to fill host variables directly through a **SELECT** statement. This method can be used only when the result consists only of one single record.

In contrast to the **SELECT ... INTO** statement, **SQLCODE** and **SQLSTATE** cannot be used to check whether a record is found. If no record is found, NULL values are returned by default. You either must use indicator variables to detect NULL values or an SQL scalar function like **COALESCE** that converts the NULL value into a default.

If the result consists of more than one row, **SQLCODE -811** is returned, but in contrast to the **SELECT ... INTO** statement, the host variables are not updated.

Example 8-5 shows how the total count of an order is calculated and returned within a **SET** statement.

Example 8-5 Results from a SET statement

```
Dcl-s Agent Packed(9);
Dcl-s TotalFlights Packed(9);

Agent = 1234;
Exec SQL
    Set :TotalFlights =
        ( SELECT COUNT(*)
          FROM ORDERS
          WHERE AGENT_NO = :Agent );
```

8.3.2 Dynamic

Dynamic SQL allows you to define an SQL statement at run time. A SQL statement can either be built by the application or the application can accept a SQL statement that is passed into it from another source. The program, and by extension the DBMS, does not know about the SQL statement before it is built. When it is processing dynamic SQL, the program is required to perform the following tasks:

- ▶ Build or access from another source a SQL statement.
- ▶ Prepare the SQL statement for processing.
- ▶ Run the SQL statement.
- ▶ Handle any SQL messages as they come up.

To issue a dynamic SQL statement, use the statement with either a **PREPARE** then **DECLARE CURSOR**, **PREPARE** then **EXECUTE** an SQL statement, or an **EXECUTE IMMEDIATE** statement because dynamic SQL statements are prepared at run time, not at precompile time. The **EXECUTE IMMEDIATE** statement causes the SQL statement to be prepared and run dynamically at program run time.

There are two basic types of dynamic SQL statements: **SELECT** statements and non-**SELECT** statements. Non-**SELECT** statements include such statements as **DELETE**, **INSERT**, and **UPDATE**.

Client/server applications that use interfaces such as Open Database Connectivity (ODBC) typically use dynamic SQL to access the database.

Suppose that you have a business analyst that works for the FLIGHTIBM i company and they need a report that allows them to select one or more pieces of information to filter the data onto the report: FLIGHT_NO, CUSTOMER_NO, CLASS, and AGENT_NO. In any single run of the report, you do not know which combinations of filter options they chose or if they chose all of them.

You might construct a dynamic SQL statement like what is shown in Example 8-6.

Example 8-6 Example of a dynamic SQL statement

```

Dcl-c quote Const('');
Dcl-s SQLStatement Char(2000);
Dcl-s WhereClause Char(2000);

SQLStatement = 'SELECT FLIGHT_NO, CUSTOMER_NO, +
                CLASS, AGENT_NO +
                FROM CUSTORDR';
If FLIGHT_NO > *ZEROS;
    WhereClause = ' Where FLIGHT_NO = ' + inflight_NO;
Endif;

If CUSTOMER_NO > *zeros ;
    If WhereClause <> *blanks;
        WhereClause = trim(WhereClause) +
            ' and CUSTOMER_NO = ' + inCUSTOMER_NO;
    Endif;
Else;
    WhereClause += ' Where CUSTOMER_NO = ' +
        inCUSTOMER_NO;
Endif;

If CLASS <> *blanks ;
    If WhereClause <> *blanks;
        WhereClause = trim(WhereClause) +
            ' and CLASS = ' + quote + inCLASS + Quote;
    Endif;
Else;
    WhereClause += ' Where CLASS = ' + quote + inCLASS +
        Quote;
Endif;

If AGENT_NO > *zeros ;
    If WhereClause <> *blanks;
        WhereClause = trim(WhereClause) +
            ' and AGENT_NO = ' + inAGENT_NO;
    Endif;

```

```

Else;
    WhereClause += ' Where AGENT_NO = ' + in AGENT_NO;
Endif;

SQLStatement = %trim(SQLStatement) + WhereClause;

Exec sql
    Execute immediate :SQLStatement;

```

Because the string is created at run time, host variables are not necessary and cannot be used. They can be directly integrated into the string. However, there are some situations where you want to use variables. In these cases, you can use a question mark (?) as a parameter marker that can be set in the **EXECUTE** or **OPEN** statement.

To convert the character string that contains the SQL statement to an executable SQL statement, one of the following actions is necessary:

► **EXECUTE IMMEDIATE:**

A string is converted to an SQL statement and run immediately. This statement can be used only if no cursor is needed.

► **PREPARE and EXECUTE:**

A string is converted and later run. Variables can be embedded as parameter markers and replaced in the **EXECUTE** statement. **EXECUTE** can be used only if no cursor is needed.

► **PREPARE and DECLARE CURSOR:**

A string is converted and the converted SQL statement is used to **DECLARE** a cursor. Like in static SQL, either a serial or a scroll cursor can be used.

If you use a variable **SELECT** list, an SQL Descriptor Area (SQLDA) is required where the returned variables are described.

EXECUTE IMMEDIATE statement

With **EXECUTE IMMEDIATE**, the command string is converted and run in a single SQL statement. It is a combination of the **PREPARE** and **EXECUTE** statements. It can be used to prepare and run SQL statements that do not contain host variables or parameter markers.

When an **EXECUTE IMMEDIATE** statement is run, the specified statement string is parsed and checked for errors. If the SQL statement is not valid, it is not run and the error condition that prevents its execution is reported in the stand-alone **SQLSTATE** and **SQLCODE**. If the SQL statement is valid, but an error occurs during its execution, that error condition is reported in the stand-alone **SQLSTATE** and **SQLCODE**. Additional information about the error can be retrieved from the SQL Diagnostics Area (or the SQLCA).

In Example 8-7, the Order table with departure date from the previous year and the appropriate Customer rows are saved in history tables. The names of the history tables are dynamically built. The year of the stored order departure data is part of the table name. In the create table statement, the table is built and filled with the appropriate data.

Example 8-7 Example of an EXECUTE IMMEDIATE statement

```

Dcl-s PrevYear Packed(4);
Dcl-s SQLString Varchar(32767);

PrevYear = %subDt(%date() : *Years) - 1;
MyFile   = 'ORDERD' + %Char(PrevYear);

```

```
SQLString = Create Table ' + %trim(mylib) + '/' +
            %trim(MyFile) + 'as (Select d.* +
            From ORDERS
            Where year(DEPARTURE_DATE) = ' +
            %char(PrevYear) + ') with data ';
Exec SQL
            Execute immediate :SQLString;
```

Combining the SQL statements PREPARE and EXECUTE

If a single SQL statement must be run repeatedly, it is better to prepare the SQL statement once by using the SQL **PREPARE** statement, and run the statement several times with the SQL **EXECUTE** statement.

When you are using the **EXECUTE IMMEDIATE** statement, the **PREPARE** statement is performed every time. This can lead to performance degradation.

Although a statement string cannot include references to host variables, it might include *parameter markers*, which can be replaced by the values of host variables when the prepared statement is executed. A parameter marker is a question mark (?) that is used where a host variable can be used if the statement string were a static SQL statement.

The PREPARE statement

The **PREPARE** statement creates an executable form of an SQL statement from a character-string form of the statement. The character-string form is called a statement string, and the executable form is called a prepared statement.

The following snippet illustrates the parts of the **PREPARE** statement that are necessary to convert a character string into a SQL string:

```
PREPARE statement-name
      FROM host variable
```

Where:

- ▶ **Statement name**
Specifies the name of the SQL statement. The statement name must be unique in your source member.
- ▶ **Host variable**
Specifies the character variable that contains the SQL string.

The EXECUTE statement

The **EXECUTE** statement runs a prepared SQL statement without a cursor.

The following snippet shows the **EXECUTE** statement:

```
EXECUTE statement_name
      (USING HostVariable1, HostVariable2,.....HostVariableN)
```

Where:

- ▶ **Statement name**
Specifies the name of the SQL statement that is run.
- ▶ **USING :HostVariable**
If parameters are used, **USING** and the host variables that contain the values must be listed. The host variables must be listed in the sequence that they are needed.

Example 8-8 shows an **INSERT** statement that is being dynamically built and then run several times.

Example 8-8 Example of using the PREPARE and EXECUTE statements

```
Dcl-s PrevYear Packed(4);
Dcl-s SqlString Varchar(32767);
Dcl-ds AgentDs ExtName(AGENTS) Dim(3) End-ds;

  For index = 1 to %elem(AgentDS);
    AgentDS(Index).AGENT_NO = Index;
    AgentDS(Index).AGENT_NAME = 'JOHN SMITH' + %CHAR(Index);
  Endfor;

  SQLString = 'Insert into ITSOLIB/AGENTHST ' +
    'Values ?, ?)';

  Exec sql prepare MyDynSQL from :SQLString;

  For index = 1 to %elem(AgentDS);
    Exec sql
      Execute SQLString
      using :AgentsDS(Index).AGENT_NO,
          :AgentsDS(Index).AGENT_NAME);
  Endfor;
;
```

Combining the SQL PREPARE and DECLARE statements

If a cursor must be defined, but the SQL **SELECT** statement cannot be defined at compile time, the character string can be built at run time. The **PREPARE** statement converts the character string to an SQL string.

The **DECLARE** statement defines the SQL cursor by using the executable SQL string instead of a **SELECT** statement.

The PREPARE statement

The **PREPARE** statement creates an executable form of an SQL statement from a character-string form of the statement. The character-string form is called a statement string, and the executable form is called a prepared statement.

The following text shows the parts of the **PREPARE** statement that are necessary to convert a character string into a SQL string:

```
PREPARE statement_name
  FROM host variable
```

Where:

► **Statement name**

Specifies the name of the SQL statement. The statement name must be unique in your source member.

► **Host variable**

Specifies the character variable that contains the SQL string. The character string can contain the **FOR READ/FETCH ONLY** clause, the **OPTIMIZE** clause, the **UPDATE** clause, and the **WITH Isolation Level** clause.

The DECLARE CURSOR statement

The **DECLARE** statement defines the cursor for the executable SQL statement. As with static SQL, a serial and a scroll cursor can be created.

Example 8-9 shows the **DECLARE CURSOR** statement and how it can be used with dynamic SQL.

Example 8-9 Example showing the usage of PREPARE and DECLARE CURSOR statements

```
Dcl-s StartDate Date;
Dcl-s EndDate Date;
Dcl-s AgentNo Zoned(9);
Dcl-s OrderTotal Zoned(11);
Dcl-s SQLString Varchar(1000);

      StartDate = D'2013-01-01';
      EndDate   = D'2013-03-31';

      SQLString  = 'Select AGENT_NO, COUNT(*) as +
                    TOTAL_ORDERS +
                    from ORDERS +
                    Where DEPARTURE_DATE between +
                        Date(''+%char(StartDate)+'') +
                        and Date(''+%char(EndDate)+'') +
                    Group by AGENT_NO +
                    Order by COUNT(*) desc';

      Exec sql prepare MySQLStmt from :SQLString;

      Exec sql Declare TotAgntCS Cursor WITH HOLD for
                    MySQLStmt;

      Exec sql Open TotAgntCs;

      Dou SQLSTATE = '02000';

      Exec SQL
          Fetch next from TotAgntCS
              into :AgentNo, :OrderTotal;

      If SQLState = '02000' or SQLCode < *zeros;
          Leave;
      Endif;

      Enddo;

      Return;
```

8.3.3 Security

As a business grows and your applications expand beyond the four walls of your building, security becomes more important. There are people outside your walls who might try to inject a virus or other destructive commands into your system just to prove that they can do it. One method of trying to get into your system is called *injection attacks*.

An injection attack is an attack that comes from input from the user and has not been checked to see that it is a valid SQL statement. The objective is to fool the database system into running malicious code. The code that is run compromises the DBMS or allows sensitive information to be revealed to the people who initiated the attack.

There are two types of injection attacks.

- ▶ First-order attacks

First-order attacks give the attacker immediate results either by a direct response from the DBMS or by an email that is sent from the system they attacked.

- ▶ Second-order attacks

Second-order attacks are when the attacker implants something (code or data) to be run later. The attack is not immediately apparent until the code is run.

There are two primary ways to mitigate injection attacks effectively:

- ▶ Ensure that your database is secured.

You should limit any user and application to the security it needs to run and not configure your system with ALL object authority. For example, financial data should not be accessed by anyone unless they are going through an authorized financial application and the application should be configured so that it can access only the data that is needed and not ALL data. One method of doing this task is to use views. Your applications should access data through views and should not directly access base tables. Views offer a level of security. Your views can be in one library and your base tables and indexes can be stored in a different library to mask where your data is stored. Also, your views should have only the columns that are needed by the program. For example, a view can give access to Employee Name from a human resource table without including payroll information. You now have secured your data from both human queries and injection attacks.

- ▶ Use parameter markers for SQL statements instead of embedding data into the formatted statement.

Parameter markers allow you to define exactly when and what type of data is passed into your query. Therefore, injection attacks that try to insert data into your SQL query are rejected unless they exactly match the data type that is defined.

Parameter markers

Parameter markers are placeholders for substituting data. Parameter markers are identified by a question mark (?) in an SQL statement. Data is substituted for each question mark in the order they appear when the SQL statement is run or the cursor is opened.

Parameter markers help dynamic SQL statement performance by allowing the query optimizer to reuse the previous execution's access plan.

Here are the steps for specifying and using a parameter marker:

1. Define an SQL statement:

```
MYSQLStatement = 'Select CUSTOMER_NAME from CUSTOMER where CUSTOMER_NO = ?';
```

2. Prepare the SQL statement for execution:

```
Exec sql prepare SQLstmt from :MYSQLStatement;
```

3. If you are using Execute, execute an SQL statement by substituting data:

```
Exec SQL Execute SQLstmt using :parameter_field_A;
```

4. If you are using a cursor:
 - a. Define the cursor:


```
Exec SQL declare Cursor-A cursor for SQLstmt;
```
 - b. Open the cursor:


```
Exec SQL open Cursor-A using :parameter_field_A;
```

8.3.4 SQL data access interfaces

Embedding SQL statements into your programs offers a powerful way of accessing data from DB2 for i. The advantages that you gain from embedded SQL far outweigh traditional I/O and RLA. Here are some of the major advantages to using embedded SQL:

► Flexibility

This is the ability to respond quickly to changing business requirements. SQL allows you to separate your programs from the underlying data model. If structured correctly, it allows you to make database changes independently of program changes. You also must touch only the programs that need the changes to be made to the table. Programs that do not care about the change and still access the data in the table do not have to be changed or recompiled.

► Scalability

SQL scales much better than RLA. As more data is processed through a specific job, RLA becomes less and less efficient. This inefficiency is contrary to the design of SQL. For programs that are written to access the data using SQL, they become more efficient as more data is processed through your job. The performance chart (shown in Figure 8-7) shows how RLA and SQL data access starts off almost even in performance. However, as more data is processed through a specific program, RLA response times start to increase while SQL access remains relatively stable.

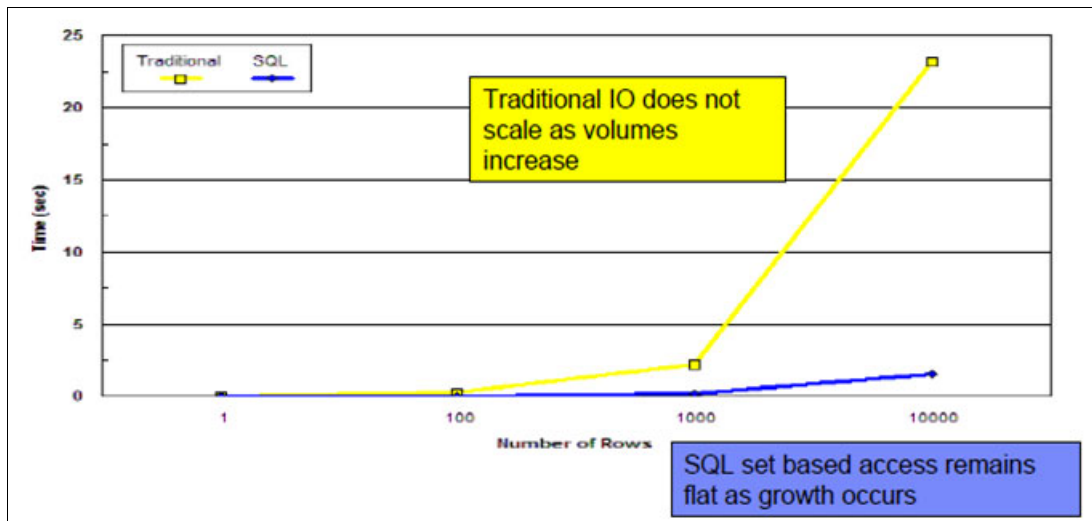


Figure 8-7 Performance comparison between RLA and SQL access

► Future direction

IBM continues to enhance SQL and SQL-based data access while putting few resources into enhancing traditional I/O. Although it is true that there are additional options using traditional I/O, the underlying RLA routines are not enhanced or optimized for today's volumes of data. The future direction of industry in general and IBM specifically is in SQL. The more you can adopt a good SQL architecture, the more you benefit from future enhancements.

► Industry standard

Because SQL is used industry-wide by all other database management systems' DBMSs, using SQL gives you the ability to more easily port data and routines back and forth from other DBMSs to DB2 for i. Stored procedures that are written for other DBMSs that follow ANSI standards can run on the IBM i.

Getting started

Before you look at specific ways in which you can access data by using SQL embedded into your programs, take a quick look at what not to do.

When they first start using SQL that is embedded into their programs, many programmers still think in terms of RLA. For example, some programmers normally structure a program in this way:

```
SETLL (ORDNO)ORDERS;
READ ORDERS;
DOW NOT %EOF(ORDERS);
  CHAIN (CUSTNO)CUSTOMERS;
  CHAIN (AGENT_NO)AGENTS;
  . . .
  READ ORDERS;
ENDDO;
```

Instead, consider structuring your first embedded SQL program this way:

```
Exec sql declare cursor-a cursor for
  select * from orders where ORDNO >= :ORDNO;
Exec sql open cursor-a;
Fetch next from cursor-a into ORDERDS;
Dow sqlcode = 0;
  Exec sql Select * into :DSCustomers
    from customers were CUSTNO = :CUSTNO;
  Exec sql Select * into :DSAgents
    from AGENTS where AGENT_NO = :AGENT_NO;
  . . .
  Exec sql Fetch next from cursor-a into :ORDERS;
Enddo;
Exec sql Close Cursor-a
```

As you can see, the SQL program is structured the same way as the traditional I/O program, except that it uses SQL statements. When this code is in production, it is possible that users complain about how slow it runs and how SQL is not as fast as traditional I/O. Yet performance issues between the two programs have nothing to do with SQL and everything to do with how the SQL program is structured.

Traditional I/O programs think in terms of one row at a time processing. SQL programs must be structured to think in terms of *set*-based processing.

Host variables

You can use different values in your SQL statement instead of literal values to make your application more flexible and to process different rows in a table. SQL allows you to embed such variables, which are called *host variables*. A host variable in an SQL statement must be identified by a preceding colon (:).

You can differentiate between the following items:

- ▶ Single host variable
- ▶ Host structure
- ▶ Host structure array

Single field host variable

A host variable is a single field in your program that is specified in an SQL statement, usually as the source or target for the value of a column. Every field that is defined in your source code can be used as a host variable. It makes no difference if the fields are defined in the File, Definitions, or Calculations Specifications. The host variable and column must be data type compatible.

Note: Array elements cannot be used as host variables.

Host variables are commonly used in SQL statements in these ways:

- ▶ In a WHERE clause of a **SELECT** statement
- ▶ As a receiving area for column values
- ▶ As a listed column in a **SELECT** statement
- ▶ As a value in other clauses of an SQL statement:
 - In the SET clause of an **UPDATE** statement
 - In the VALUES clause of an **INSERT** statement
 - As parameters in the SQL **CALL** statement
- ▶ Using host variables in a WHERE clause

When using host variables in a WHERE clause, you can use a host variable to specify a value in the predicate of a search condition, or to replace a literal value in an expression.

Suppose you want to know what customers are flying in the month of July. Your code might look like what is shown in Example 8-10.

Example 8-10 Example using a WHERE clause

```
Dcl-s StartDate date inz(D'2013-07-01');
Dcl-s EndDate   date inz(D'2013-07-31');

Exec SQL
      Select CUSTOMER_NO into :HstCustNo
      From ORDERS
      Where DEPARTURE_DATE between :StartDate   and
                                   :EndDate;
```

- ▶ Using Host variables as a receiving area for column values

When you use Host variables as a receiving area for column values, the Host variables can be used in an **SELECT ... INTO** statement or FETCH clause to retrieve values from the return table. Example 8-11 shows how to get the total number of flights an agent has sold.

Example 8-11 Example using a host variable in a SELECT...INTO statement

```
Dcl-s Agent Packed(9);
Dcl-s TotalFlights Packed(9);
```

```

Agent = 1234;
Exec SQL
    SELECT COUNT(*)
    INTO :TotalFlights
    FROM ORDERS
    WHERE AGENT_NO = :Agent;

```

► Using host variables as a value in a **SELECT** statement

When you are using host variables as a value in a **SELECT** statement, and when you are specifying a list of items in the **SELECT** statement, you are not restricted to the column names of tables and views, as shown in Example 8-12.

Example 8-12 Example using a host variable in a SELECT clause

```

Dcl-s Agent Packed(9);
Dcl-s TotalFlights Packed(9);
Dcl-s Commission$ Packed(5:2);

Agent = 1234;
Commission$ = 50.00;
Exec SQL
    SELECT COUNT(*) * :Commission$ as
           Commission
    INTO :TotalFlights
    FROM ORDERS
    WHERE AGENT_NO = :Agent;

```

► Using host variables in the VALUES clause of an **INSERT** statement

When using host variables in the VALUES clause of an **INSERT** statement, a row can be inserted by using host variables in the VALUES clause, as shown in Example 8-13.

Example 8-13 Example using the host variable in the VALUES clause of an INSERT statement

```

Dcl-s AgentNo Packed(9);
Dcl-s AgentName Varchar(25) Inz('John Smith');

Exec SQL
    Insert into agents
           (AGENT_NO, AGENT_NAME)
    Values (:AgentNo, :AgentName);

```

► Using host variables as parameters in the **CALL** statement

When you are using host variables as parameters in the **CALL** statement, parameters can be passed as host variables when calling a stored procedure, as shown in Example 8-14.

Example 8-14 Example using a host variable in a CALL statement

```

dcl-s AgentNo Packed(9);

Exec SQL
    Call GetAgentByName ( :AgentNo );

```

Host structure

A host structure is a group of host variables that are used as the source or target for a set of selected values (for example, the set of values for the columns of a row). A host structure is defined as an internally or externally described data structure in your source code. Host variables are commonly used in SQL statements as a receiving area for column values.

Host structures can be used in an **SELECT ... INTO** statement or a **FETCH** clause. The **INTO** clause names one or more host variables that you want to contain column values that are returned by SQL, as shown in Example 8-15.

Note: When you are using host variables for each variable, a separate pointer must be returned. When you are using host structures, only one pointer is returned. That single pointer might help create a performance gain.

Example 8-15 Example of a host structure in a SELECT...INTO statement

```
Dcl-ds AgentDS;
  AGENT_NO    Packed(9);
  AGENT_NAME  VarChar(35);
End-ds;

      Select AGENT_NO, AGENT_NAME
             Into :AgentDS
             FROM AGENTS
             Where AGENT_NO = 12345;
```

Host structure array

A host structure array is defined as a multi-occurrence data structure or an array data structure. Both types of data structures can be used on the SQL **FETCH** or **INSERT** statements when processing multiple rows. The following list of items must be considered when using a data structure with multiple row blocking support:

- ▶ All subfields must be valid host variables.
- ▶ All subfields must be contiguous. The first **FROM** position must be 1, and there cannot be overlaps in the **TO** and **FROM** positions.
- ▶ If the date and time format and separator of date and time subfields within the host structure are not the same as the **DATFMT**, **DATSEP**, **TIMFMT**, and **TIMSEP** parameters on the **CRTSQLRPGI** command (or in the **SET OPTION** statement), the host structure array is not usable.

Blocked **FETCH** and blocked **INSERT** are the only SQL statements that allow an array data structure. A host variable reference with a subscript like `MyStructure(index).MySubfield` is not supported by SQL.

Note: Using blocked processing brings performance advantages because only one single pointer must be returned for a group of rows.

Using host variables

Host variables are commonly used in SQL statements in these ways:

- ▶ As a receiving area for column values to receive multiple rows in a single fetch, as shown in Example 8-16.

Example 8-16 Example of using a receiving area for multiple rows in a single fetch

```
Dcl-ds AgentsDS      ExtName(AGENTS)
                    Qualified Dim(3) End-ds;
Dcl-s Elements Uns(3) Inz(%Elem(AgentsDS));
Dcl-s Index Uns(3);

Exec SQL
Declare AgentCS Cursor for
Select AGENT_NO, AGENT_NAME
From AGENTS
Where AGENT_NO > 12345
For read only
Optimize for 100 rows;

Exec SQL open AgentCS;

Exec SQL
Fetch next from AgentCS
For :Elements rows
Into :AgentsDS;

For Index = 1 to %Elem(AgentsDS);
    Dsply AgentsDS(Index).AGENT_NAME;
Endfor;
```

- ▶ To insert multiple rows into a table (see Example 8-17).

Example 8-17 Example of inserting multiple rows in a table

```
Dcl-ds AgentsDS      ExtName(AGENTS)
                    Qualified Dim(3) End-ds;
Dcl-s AgentNo Zoned(9) Inz(12345);
Dcl-s Elements Uns(3) Inz(%Elem(AgentsDS));
Dcl-s Index Uns(3);

Clear AgentsDS;

For Index = 1 to %Elem(AgentsDS);
    AgentsDS(Index).AGENT_NO = Index;
    AgentsDS(Index).AGENT_NAME = 'John Smith';
Endfor;

Exec sql
Insert into AGENTS
:Elements Rows
Values (:AgentsDS);
```

Naming considerations for host variables

Any valid ILE RPG for IBM i variable name can be used for a host variable with the following restrictions:

- ▶ Do not use host variable names or external entry names that begin with the characters 'SQ', 'SQL', 'RDI', or 'DSN'. These names are reserved for the database manager.
- ▶ The length of host variable names is limited to 64, except in RPG, where field names can be defined with up to 4096 characters.
- ▶ The names of host variables should be unique within the source code of the member. If the same name is used for more than one variable and the data types are different, the data type of the host variable is unpredictable. Consider this situation when you are using local field definitions in RPG procedures. You do not have to define your host variables as global variables, but the names should be unique in your source.

If a data structure has the **QUALIFIED** keyword, the subfields of that data structure can have the same name as a subfield in a different data structure or as a stand-alone variable. The subfield of a data structure with the **QUALIFIED** keyword must be referenced by using the data structure name to qualify the subfield name.

Putting it all together

Consider some options to structuring your SQL program. The first thing that you want to do is determine what data you need. Traditionally, you can accept the entire record format into your program by completing the following steps:

1. For SQL-based access, you want to bring only data that is required in to your program. For our example, we need ORDER_NO, CUSTOMER_NO, CUSTOMER_NAME, AGENT_NO, and AGENT_NAME. The program is going to need only those orders in third class and only the flights that are coming in the future. We do not care about flights for today or that are completed.
2. Create a view that defines what the program needs. This view is used to define a data structure in your program and the source of your data. The following example shows how to create a view that is joining the needed columns:

```
CREATE VIEW FUT_FLTS_V as ( SELECT  ORDER_NO, CUSTOMER_NO , CUSTOMER_NAME,
AGENT_NO, AGENT_NAME, CLASS
FROM ORDERS
INNER JOIN  CUSTOMERS
ON ORDERS.CUSTOMER_NO = CUSTOMERS.CUSTOMER_NO
INNER JOIN  AGENTS
ON ORDERS.AGENT_NO = AGENTS.AGENT_NO
WHERE DEPARTURE_DATE > CURRENT_DATE);
```

Note: You could have added a condition in the view to include only third-class travelers. However, that means you might have to have a view for every class of traveler. In this case, it makes sense to filter out all past flights in the view but allow the program to determine what class of traveler to process.

3. Define an external data structure that is based on our view and a multiple occurrence data structure to hold an entire set of data (as shown in the following code). These variables are known as host variables or host data structure variables.

```
Dcl-ds Flightds Extname(FUT_FLTS_V) End-ds;
Dcl-ds Flightds Likeds(Flightds)
Occurs(32767);
```

Note: Consider the following items:

- ▶ All host variables that are used in SQL statements must be explicitly declared. LOB, ROWID, and binary host variables are not supported in IBM RPG/400®.
- ▶ Host variables are used in SQL statements by placing a colon (:) before the host variable.
- ▶ You want to set the Occurs limit as high as possible given the memory limits of your machine and the limits inherent in the language used. For the preceding example, the limits on the Array size (Occurs) is 32767 or 16 MB in size. This is one reason to bring in only the columns that your program needs and *not* the entire record format. Reducing the amount of data to only what your program needs allows you to bring in more data at a time, improving performance and efficiency of your program.

4. Next, define indicator variables. An *indicator variable* is a halfword integer variable that is used to communicate additional information about its associated host variable. Indicator variables are a special host variable that are used to allow RPG programs to check for NULL values being passed into your program during a **SELECT INTO** or **FETCH** statement. The following example defines the indicator variables:

```
Dcl-ds NULLIND;
  ORDERIN   Int(5);
  CUSTNOIN  Int(5);
  CUSTNMIN  Int(5);
  AGENTNOIN Int(5);
  AGENTNMIN Int(5);
  ArrInd    Int(5) Dim(5) Pos(1);
End-ds;
Dcl-ds NULLINDSDS Likeds(NULLIND) Dim(32767);
```

Note: You can use an array for the null indicators. The individual element can be referenced by an index (for example, NULLIND DIM(5) if nullind(3) = *Null;.

5. Define your **CURSOR** and **SELECT** statements and then open the cursor for reading, as shown in the following example:

```
Exec SQL
  Declare FutureFlight_Cursor CURSOR for
  SELECT ORDER_NO, CUSTOMER_NO , CUSTOMER_NAME,
         AGENT_NO, AGENT_NAME
  FROM   FUT_FLTS_V
  WHERE  CLASS = '3'
  FOR READ ONLY;
exec sql
  open FutureFlight_Cursor;
```

Note: The FOR READ ONLY tells DB2 for i that it does not need to lock the records that are read. This setting helps the DBMS to perform faster.

6. The code in Example 8-18 on page 253 fetches 32767 rows at a time from the cursor (defined by occurrences). It shows how to get the number of rows read and loop through the multiple occurrence data structure performing whatever action is needed. At the end of the loop, it reads the next 32767 rows until it runs out of records in the result set.

Example 8-18 Example of fetching 32767 rows at a time from the cursor

```
sqlcode      = *zeros;
LastRow = *Zeros;

//-----
//Loop until no more records to process
dow sqlcode = 0 and
    LastRow = *zeros;

exec sql
    fetch next
    from FutureFlight_Cursor
    for 32767 rows
    into :Flightsds :NULLINDSds;
```

Note: For each row inserted in Flightsds data structure the NULLINDSDS data structure has a row that will tell you additional information about the data being read (for example if a specific column is NULL).

```
//-----
//Get # of rows fetched and last row indicator
exec sql
get diagnostics :rowcount = ROW_COUNT,
                :lastrow = DB2_LAST_ROW;
for CurrentRecord = 1 to rowcount ;
%occur(Flightsds) = CurrentRecord;
%occur(NULLINDSds) = CurrentRecord;
```

Note: You reference a field from the current occurrence of the data structure by datastructurename.fieldname. For example to reference CUSTOMER_NAME, you would use FLIGHTSDS.CUSTOMER_NAME. To see whether CUSTOMER_NAME contains NULLs, use NULLINDSDS.CUSTNMIN.

```
... do something ...

next;
enddo;
exec sql
    close FutureFlight_Cursor;
```

Deleting in sets

There are different ways to delete data using a set-based architecture. The simplest is to run a **DELETE** statement as follows:

```
DELETE FROM table_a where table_a_column_a = :somehostvalue;
```

However, you might also want to delete from table_a based on rows in table_b:

```
DELETE FROM table_a where exists (SELECT * from table_b where
                                Table_b_column = table_a_column);
```

Inserting in sets

Inserting in sets (bulk) is the most efficient way to add multiple rows to a table and there are different ways to get the job done.

- ▶ If you have rows that are loaded into a data structure array, you can insert them into a table by using the following code:

```
Exec sql
insert into table_a
      :array_rows rows values (:table_datastructure);
```

In this case, `array_rows` is a host variable that defines the number of rows that are being inserted at a time and `table_datastructure` is an externally defined data structure that is based on either the base table definition or a view.

- ▶ If you want to insert into `table_a` based on rows in `table_b`, you might use the following code:

```
Exec sql
      insert into table_a
      select * from table_b;
```

Note: This method is equivalent to CPYF, but takes into account SQL column definitions such as Identity Columns.

Updating in sets

The easiest form of updating is changing a value in a field for all rows in the table:

```
Exec sql
      Update table_a
      set column_a = somevalue;
```

However, suppose you want to update a value in `table_a` based on if a row exists in `table_b`:

```
Exec sql
      update table_a
      Set column_a = somevalue
      Where exists (select * from table_b
      Where table_a_keyfield = table_b_keyfield);
```

Note: Joins are not valid on **UPDATE** statements. Use a **WHERE** clause to either check for the existence of a row or to join rows together from different tables. You can also consider the usage of an 'Instead of Trigger' to make an SQL view that is "read only" updatable.

Now, suppose you want to update a value in `table_a` with a value from `table_b`. You still need to check the existence of the row in `table_b`. Otherwise, `table_a` might be updated incorrectly.

```
Exec sql
      Update table_a
      Set column_a = (select column_a from table_b
      where table_a_keyfield = table_b_keyfield)
      Where exists (select * from table_b
      Where table_a_keyfield = table_b_keyfield);
```

Summary

Set-based thinking takes some time to understand before it becomes natural for structuring your programs. However, it is powerful and efficient. For every row that is read by a **CHAIN** command from a Logical File, three I/Os occur to the disk. Compare that to a **SQL SELECT** statement reading from an **INDEX**, where there are times SQL can get all of the data directly from the **INDEX** without ever needing to go to the base table. SQL is optimized and remains efficient even as the amount of data your business processes grows.

Over time, SQL continues to be enhanced with new features and performance gains. There should be few exceptions where traditional I/O is needed. When RLA is used, the following things are sacrificed:

- ▶ Flexibility
- ▶ Scalability
- ▶ Industry standard
- ▶ Future direction

8.3.5 Preferred practices from a database perspective

The goals and criteria governing preferred practices from a database perspective differ greatly from the goals and criteria governing preferred practices from an application programming perspective. For example, criteria governing preferred practices from an application programming viewpoint focus mainly on internal code and readability. This focus includes topics such as how structured the program is, whether the design of the program is using a modular approach to its architecture, or whether the program code is indented and has a good use of subroutines, procedures, comments. However, criteria for preferred practices from a database perspective focus on performance, flexibility, scalability, and so on.

From a DB perspective, the main question you want to ask as you design your program is: "What happens to the program if one of the tables that is being used changes (a column is added, modified, deleted) and you do not need or care about the change that is being made?" If the answer is, "The program must be recompiled or it abnormally ends with Level Check", you might want to reconsider the design and structure of your code.

Although individual programmers can and do vary from preferred practices, the following list of preferred practices gives you the best outcome for performance, usage of system resources, flexibility, and scalability of your application over time:

- ▶ Do not use **SELECT * FROM** against a base table in a SQL statement.

This is an industry standard that spans DBMS and it is not dependent on DB2 for i. When using **SELECT *** in your SQL statement, the DBMS rewrites your statement to include all columns from the table. If your database engineer adds a column to the table, your program receives this new column regardless of whether you need the new column. When your program receives this new column into a data structure that is not expecting the data, your program generates a SQL exception error.

Solution: Your program should read from a view that has a list of the columns that are needed. The view should be specific to your program so that if changes to the view are necessary, no other program is affected. You now have achieved separation from the physical data model and independence from all other application programs.

- ▶ Do not open SQL objects (Tables / Indexes / Views) in F specs in your program.

At first, this might seem like a strange standard to have. However, if your program abnormally ends when a change is made to a SQL table, then you must rethink the structure of your program. Opening your files in F specs checks the current structure of the file against the structure of the file that is compiled into the program. If they are different, a format level check is generated. Setting level check off does not help. You are bringing in a record layout that differs from your program. Creating an index that has only the columns your program needs helps prevent level checks. However, this is the same as using a view for data access with the exception that the index must be maintained by the system every time a row is inserted or changed, causing your system to work harder than it needs to.

Solution: You want separation of your program from the underlying data model and you want to move away from format level checks. Ideally, you do not want any data-related F specs in your program at all. All data access should come through the logical layer of the data model, which means either calling a data access stored procedure or running a SQL **SELECT** statement against a view.

Sometimes, you must have a data-related object (TABLE, INDEX) in an F spec so that the usage is recognized by an impact analysis tool such as analysis. This is especially true when using dynamic SQL statements where the FROM clause and file name might not show up as being used in your program. In these cases, put the table or index in your F spec with a USROPN clause. In your code, use a statement similar to the following one:

```
Ffile_a    if e          k disk    usropn

IF *INLR AND NOT *INLR;
  OPEN file_a;
ENDIF;
```

The above statement allows you to see your file being used by this program in an impact analysis tool, but it still gives you the benefit of *not* having to worry about format level checks. You have the best of both worlds. You can modify the underlying data model without having to worry about programs that do not care about your changes, and your impact analysis tools continue to give you accurate information.

You can also use the **TEMPLATE** keyword in an F spec to designate that this file is for reference only. You do not need to code I/O operations for the file when the **TEMPLATE** keyword is used.

Here is a code snippet where you use the **TEMPLATE** keyword:

```
FrcdFile_t IF E          K DISK    TEMPLATE
F                                     EXTDESC('EMPADDRESS')
The DSPPGMREF command returns the following:
Object . . . . . :      EMPADDRESS
  Library . . . . . :      *LIBL
  Object type . . . . . :      *FILE
```

- ▶ Use views between your program and the underlying base table. Do not access the SQL Table directly.

This has been a common theme throughout the first two preferred practice statements. Define a view with the columns that your program needs. The view should be specific to your program. Use the view as an external data structure. Access the data by using a **SELECT column_a, column_b** from *view_a*.

If a new column is added to the table and your program does not need or care about the new column, your program does not need to be changed or recompiled.

If an existing column is deleted from the table and your program is accessing the column through the view, the view can be changed to account for the deleted column. Either the view is changed to substitute blanks or the view is changed to get the data from a different table.

If an existing character-based column is increased in size and your program does not need the extra characters, the view can be changed to force the smaller size by substringing the existing column.

The point is to structure your code so that it is least likely to have issues when changes are made to the underlying tables. Move away from format-level checks.

- ▶ Every SQL statement is analyzed by the DB2 for i Optimizer to determine index needs.

In a data-centric world, indexes are used to help DB2 for i access data when a SQL statement is run. With few exceptions, indexes are *not* meant to be accessed directly within your programs.

The index advisor offers outstanding assistance in determining indexing needs to assist DB2.

- ▶ Use the DATE data type instead of NUMERIC(8,0) or character fields.

It is not uncommon to find dates that are stored as NUMERIC(8,0), DECIMAL(7,0), or CHAR(xx) in DDS files on IBM i. However, when defining a SQL Table, it is much better to define the date data type as DATE instead of alpha or numeric. SQL offers a significant number of DATE functions to help you convert your dates into other date formats. For example, there are functions to convert to Julian date. Date data types can be calculated to give you the number of days from the current day. The DBMS accounts for leap years so you do not have to do it. The DBMS compares dates and subtract two dates for you and give you the number of days between them. It certainly is possible for you to write these same functions. However, let the DBMS work for you. It is correct, accounts for various exceptions, and it runs faster than if you develop the functions yourself. Also, DATE data types are consistent with other DBMSs, which makes it easier for them to coexist.

- ▶ Every table should have a primary key (PK). The PK should be an identity column unless the data is static and does not change.

Primary Keys (PKs) are not as commonly used among IBM i developers. However, identity columns, which are used as PKs, offer a flexible approach to data modeling. It becomes dangerous to use business data as keys that relate tables together. As a business grows and expands, the duplication of data across tables can hinder a company's growth if the business column is too small. Because the business data is used to relate tables together, all tables that contain the same column must expand if the column must be increased. Hundreds if not thousands of programs are changed to account for one column expanding only because that column is duplicated across tables and is used as a key to relate data together.

An alternative is to use identity columns in a PK FK relationship. These columns are used to join data in your SQL statements, either embedded in programs or used in views. There should be no need to bring an identity column into your program. Therefore, if for some reason the identity column must be expanded, your programs are not affected. In this design, your business data is stored in one place and not duplicated in your system. If a piece of business data must expand, the column is expanded in one place because it is stored only in one spot.

This can be a powerful design that offers you significant flexibility and expandability over time as your business grows.

The one exception to this practice is when data is static. A table that contains the 50 states in the US would be an example of a static table. In this case, the PK might be the two character state code (also known as the natural key).

A primary key with an identity column is not required in cases where there is no parent-child relationship. However, there are other cases where you might want an identity column as a PK. For example, cases where you are replicating data to another DBMS as an Operational Data Store (ODS) can benefit enormously by using an identity column as a PK. Using this setup, you can change the data by selecting the ROW CHANGE TIMESTAMP column and then updating the ODS by the PK.

Note: Try to avoid cases where your parent table is linked to a child table (PK/FK) and your child table is a parent to another child table. This is called grandchildren. These situations tend to be confusing from a design perspective and have limitations with CASCADING DELETES and clearing tables.

- ▶ Create unique indexes for business data that must be unique. Do not create the indexes as PKs.

Here again you want the DBMS to do the work for you. If you must a piece of data be unique, create an index that enforces a unique constraint. There is no need for each program to enforce this business rule when the DBMS can do it faster, more efficiently, and more consistently across your applications than individual programs that are trying enforce this rule. If, at some point, the rule changes to allow duplicates, all you need to do is remove the **UNIQUE** keyword from the index. There should be no need to touch any of the application programs.

You can create multiple unique indexes where multiple business keys must be unique. For example, the department number in the department table is unique, but so is the department name. A unique index or constraint on both columns prevents duplication of either value. The unique business rule is coded in one place and serves all applications. This rule also helps the DBMS perform queries that involve **SELECT MAX(column_a)**, where the column is known by the DBMS as being unique. The unique constraint on the index gives the DBMS information that allows it to perform these queries much faster and more efficiently.

- ▶ Use DBMS defaults for columns whenever possible. For example, CURRENT_DATE or CURRENT_TIMESTAMP.

Push as much work to the DBMS as you can so that your programs do not have to work as hard. This process reduces the complexity of your programs, makes them easier to read, makes them more efficient because the work is done by the system, takes advantages of more options in future releases, and so on. There are many examples where you might not want to deal with a column if the data is not provided. However, you still need a default value that is inserted into a column where the row is written. Using system defaults allows you to omit the column on an insert and still have a default value entered.

- ▶ SQL statements that are embedded into your programs should be as simple as possible. Complex SQL statements should be in views and accessed by high-level language (HLL) programs.

There are many reasons why you want to keep the complexity of a SQL statement outside of your program:

- It makes your program easier to read.
- Views are component modules that can be reused.
- The same complex SQL statement that is needed in multiple programs can be maintained in one place instead of duplicating the code in each program.
- When business logic changes, you can change the view without having to touch any of the programs.

However, you want to think about performance when you are structuring your views. For example, if your program needs to **CAST()** a column to a new type, putting the **CAST** statement into a view requires that every row in the view have the **CAST** statement run when the view is queried. If the **CAST** statement is part of the **SELECT** in the program, the **CAST** is run only on the rows that are brought into your program. This can make a performance difference if you are using a **WHERE** clause in your **SELECT** statement in your program and you are bringing in a small, select number of rows to be processed. Here is a comparison of two options for casting a column:

```
CREATE VIEW VIEW_A AS (  
SELECT CAST(AGENT_NO as CHAR(10) ) AS CHAR_AGENT_NO, AGENT_NAME  
FROM AGENTS )
```

```
CREATE VIEW VIEW_B AS (  
SELECT AGENT_NO , AGENT_NAME  
FROM AGENTS)
```

If your program that executes a select against these views, as shown in the following code, the usage of **VIEW_B** and the **SELECT** from **VIEW_B** perform better:

```
SELECT * FROM VIEW_A WHERE CHAR_AGENT_NO = '1';  
OR  
SELECT CAST(AGENT_NO as CHAR(10), AGENT_NAME  
FROM VIEW_B  
WHERE AGENT_NO = 1;
```

In the first example (**VIEW_A**), every agent row requires the agent number to be converted to a character (**CAST**) before the **WHERE** clause filters agents. In the second example (**VIEW_B**), agent number 1 is returned to your query and then the number is converted to a character. This method requires only one row to have the **CAST** statement run against it.

- ▶ Always use **GET DIAGNOSTICS** at the end of every **FETCH**.

The SQL DBMS gives you a significant amount of information if you ask for it. After fetching data from a table, the **GET DIAGNOSTIC** statement returns information such as Number of Rows Fetched, if the last rows in the table were fetched (end of file), and if any SQL errors occurred. It is important to retrieve this information after each fetch or SQL **SELECT** so that your program is aware of any issues that occurred. Just as an indicator is checked after a **CHAIN** command or **%EOF(file_a)** is checked, **GET DIAGNOSTICS** gives you the information that you need to check how the **FETCH** command worked.

If **SQLCODE** is something other than zero, check **SQLSTATE**. **SQLSTATE** is the preferred standard return code. It provides application programs with return codes for common error conditions that are found among the DB2 products. **SQLSTATE** codes are useful when you are handling errors in distributed SQL applications.

- ▶ Use both short names and long names for columns. Column long names should be consistent across all tables.

Field names can have both a short name (10 characters or less) and a long name (up to 128 characters). When accessing a table through a SQL query, either the long name or the short name can be used, so the short column name can be used by the programs and the long names can be known and referenced by business users. Here is the syntax of the **CREATE TABLE** command for the columns:

```
FIELD_LONG_NAME      FOR FLDNAM      typedefinition
```

Make the short name consistent across all tables. Define a glossary of standard abbreviations and use these abbreviations to construct a meaningful short name. For example, a customer number might be defined as CUSTNUMB, CUSTNBR, or CUSTNO, where CUST, NUMB, NBR, and NO are standard or business abbreviations for customer and number. Avoid the usage of symbols in column names because they can be problematic with various development tools. It is customary to format the field name with two or three characters that represent the table name and then the rest of the field name to be the field definition. For example, a customer number field in a flight order header table is Returning Indicator Variables that are formatted as OHCUSTNO, where OH is the prefix that is used for every field name in the table ORDER HEADER.

Make the column long name consistent across all tables. For example, for customer number, the column name should be CUSTOMER_NUMBER and should be the same name regardless of what table it is in. This convention allows your business users to know the customer number by only one name and not have to worry about what the field name is for every file. It also allows the DBMS to analyze the database easier. If there is ever a need to reverse engineer your database into a modeling tool, the database relationships can be inferred more accurately by using consistent column names.

On DB2 for i, column names are stored internally in uppercase. Although customerNumber or CustomerNumber might appear to be more appealing to the eye, they can become undecipherable when debugging a program. In this case, both versions of customer number (camel case or mixed case) appear as CUSTOMERNUMBER when reviewing debug data. An underscore can help make the columns more readable.

8.3.6 Result sets

With the introduction of IBM i 7.1, RPG programs can use result sets that are produced by another RPG program. This is powerful because there are size limits for passing arrays, but no limitations for passing result sets.

Result sets are in memory. You receive a pointer that is called a RESULT_SET_LOCATOR to the memory location and you can process the result set one row at a time or multiple rows at a time. Here is the sequence of events:

1. Define a new field of type RESULT_SET_LOCATOR.

A new field type definition is available in the D specs:

```
D MyResultSetSSQLTYPE(RESULT_SET_LOCATOR)
```

2. Call the RPG program or Stored Procedure that returns a result set.

This is simply a call to a Stored Procedure or RPG program that opens a cursor and returns a result set:

```
CALL GetMyResultSets();
```

3. Associate the RESULT_SET_LOCATOR field with the returned result set.

The RESULT_SET_LOCATOR (MyResultSet) must be ASSOCIATED (connected) to the result set that is now available to your program:

```
Exec SQL
  ASSOCIATE RESULT SET LOCATORS
  (:MyResultSet)
  WITH PROCEDURE GetMyResultSets;
```

Note: If the program returns multiple result sets, you can associate multiple RESULT_SET_LOCATORS to each result set by separating the RESULT_SET_LOCATORS by commas. They each must be defined first. For example:

```
(:MyResultSets, :rsCustomers, :rsOrders)
```

4. **ALLOCATE** a **CURSOR** for RESULT_SET_LOCATOR.

Open the result set in a **CURSOR**:

```
Exec SQL
    ALLOCATE MyCursor CURSOR
    FOR RESULT SET :MyResultSet;
```

```
Exec SQL
    Open MyCursor;
```

Note: It is not necessary to explicitly open the cursor. However, it is preferred practice for readability.

5. Process data from **CURSOR**.

The following code shows how to fetch data from the cursor and process the rows:

```
Dow  SQLcode = 0
```

```
Exec SQL  FETCH next from MyCursor into MyDataStructure;
```

```
Do something to row . . .
Enddo;
```

```
Exec SQL close MyCursor;
```

Using multiple occurrence data structures or arrays

It is also possible to **FETCH** more than one row at a time into an array or multiple occurrence data structure and then process through the data structure. This is a good strategy if you must do bulk inserts, updates, or deletes from the rows in the result set. To do so, complete the following steps:

1. Define a multiple occurrence data structure or a data structure array.
2. **FETCH** into data structure for %ELEM(data structure);. For example:

```
Dcl-ds MyAgentStructure Extname(AGENTS)
        Occurs(1000) End-ds;
Dcl-s rsAGENTS SQLTYPE(RESULT_SET_LOCATOR);
Dcl-s NextRecords Zoned(9);
```

```
NextRecords = %Elem(MyAgentStructure);
GETAGENTS();
Exec SQL
    ASSOCIATE RESULT SET LOCATORS
    (:rsAGENTS)
    WITH PROCEDURE GETAGENTS;
```

```
Exec SQL
    ALLOCATE AgentsCursor CURSOR
    FOR RESULT SET :rsAGENTS;
```

```

Exec SQL
  FETCH NEXT from AgentsCursor
    for NextRecords
  into : MyAgentStructure;

```

Note: The following things must be taken into account:

- ▶ Individual rows can be processed by %occurs(MyAgentStructure) = indexnumber.
- ▶ Individual columns can be processed by MyAgentStructure(indexnumber).ColumnName.

8.3.7 Joins

Sometimes the information you want to see is not in a single table. To form a row of the result table, you might want to retrieve some column values from one table and some column values from another table. You can retrieve and join column values from two or more tables into a single row.

Several different types of joins are supported by DB2 for i:

- ▶ Inner join
- ▶ Left outer join
- ▶ Right outer join
- ▶ Left exception join
- ▶ Right exception join
- ▶ Full outer join
- ▶ Cross join

Inner join

An inner join returns only the rows from each table that have matching values in the join columns. Any rows that do not have a match between the tables do not appear in the result table.

With an inner join, column values from one row of a table are combined with column values from another row of another (or the same) table to form a single row of data. SQL examines both tables that are specified for the join to retrieve data from all the rows that meet the search condition for the join. There are two ways of specifying an inner join: using the **JOIN** syntax, and using the **WHERE** clause.

For example, start by assuming that you have a table that is named *airline* that contains a list of available airlines (Figure 8-8) and you have a table that is named *flights* of available flights for each airline (Figure 8-9 on page 263).

AIRLNM	AIR...
Airline_1	AMA
Airline_2	NWA
Airline_3	CON
Airline_4	UNI
Airline_5	DLT
Airline_6	TWA
Airline_7	DLH

Figure 8-8 Table of available airlines

AIRLINES	FLIGHT_NUMBER	DEPARTURE	DAY_OF_WEEK	ARRIVAL	DEPARTURE_TIME	ARRIVAL_TIME	SEATS_AVAIL...
AMA ...	120002	Albany	Monday	Albuquerque	09:03 AM	... 11:03 AM	... 250
AMA ...	130003	Albany	Monday	Albuquerque	10:06 AM	... 12:06 PM	... 250
AMA ...	140004	Albany	Monday	Albuquerque	11:09 AM	... 01:09 PM	... 250
AMA ...	150005	Albany	Monday	Albuquerque	12:55 PM	... 02:55 PM	... 250
AMA ...	160006	Albany	Monday	Albuquerque	02:01 PM	... 04:01 PM	... 250
AMA ...	170007	Albany	Monday	Albuquerque	03:29 PM	... 05:29 PM	... 250
AMA ...	180008	Albany	Monday	Albuquerque	05:30 PM	... 07:30 PM	... 250
AMA ...	190009	Albany	Monday	Albuquerque	07:28 PM	... 09:28 PM	... 250
AMA ...	210010	Albany	Tuesday	Albuquerque	08:01 AM	... 10:01 AM	... 250
AMA ...	220011	Albany	Tuesday	Albuquerque	09:03 AM	... 11:03 AM	... 250
AMA ...	230012	Albany	Tuesday	Albuquerque	10:06 AM	... 12:06 PM	... 250
AMA ...	240013	Albany	Tuesday	Albuquerque	11:09 AM	... 01:09 PM	... 250
AMA ...	250014	Albany	Tuesday	Albuquerque	12:55 PM	... 02:55 PM	... 250
AMA ...	260015	Albany	Tuesday	Albuquerque	02:01 PM	... 04:01 PM	... 250
AMA ...	270016	Albany	Tuesday	Albuquerque	03:29 PM	... 05:29 PM	... 250
AMA ...	280017	Albany	Tuesday	Albuquerque	05:30 PM	... 07:30 PM	... 250

Figure 8-9 Table of available flights

Now, say you want to get a list of flights available by Airline Name instead of the airline three character abbreviation. You can join the airline table with the available flights table by three character abbreviation. Your SQL statement look like the following one:

```
SELECT AIRLNM, FLIGHT_NUMBER, DEPARTURE, ARRIVAL, DAY_OF_WEEK, DEPARTURE_TIME,
ARRIVAL_TIME, SEATS_AVAILABLE
FROM AIRLINE A
INNER JOIN FLIGHTS B
ON A.AIRLIN = B.AIRLIN;
```

The results of running this SELECT against these tables are shown in Figure 8-10.

AIRLINES	FLIGHT_NUMBER	DEPARTURE	ARRIVAL	DAY_OF_WEEK	DEPART
Airline_1...	6205465	Albany	Winnipeg	Saturday	09:03 AM
Airline_1...	6305466	Albany	Winnipeg	Saturday	10:06 AM
Airline_1...	6405467	Albany	Winnipeg	Saturday	11:09 AM
Airline_1...	6505468	Albany	Winnipeg	Saturday	12:55 PM
Airline_1...	6605469	Albany	Winnipeg	Saturday	02:01 PM
Airline_1...	6705470	Albany	Winnipeg	Saturday	03:29 PM
Airline_1...	6805471	Albany	Winnipeg	Saturday	05:30 PM
Airline_1...	6905472	Albany	Winnipeg	Saturday	07:28 PM
Airline_1...	7105473	Albany	Winnipeg	Sunday	08:01 AM
Airline_1...	7205474	Albany	Winnipeg	Sunday	09:03 AM
Airline_1...	7305475	Albany	Winnipeg	Sunday	10:06 AM
Airline_1...	7405476	Albany	Winnipeg	Sunday	11:09 AM
Airline_1...	7505477	Albany	Winnipeg	Sunday	12:55 PM
Airline_1...	7605478	Albany	Winnipeg	Sunday	02:01 PM
Airline_1...	7705479	Albany	Winnipeg	Sunday	03:29 PM
Airline_1...	7805480	Albany	Winnipeg	Sunday	05:30 PM
Airline_1...	7905481	Albany	Winnipeg	Sunday	07:28 PM
Airline_4...	1105482	Albuquerque	Albany	Monday	07:33 AM
Airline_4...	1205483	Albuquerque	Albany	Monday	08:36 AM
Airline_4...	1305484	Albuquerque	Albany	Monday	09:38 AM
Airline_4...	1405485	Albuquerque	Albany	Monday	10:41 AM
Airline_4...	1505486	Albuquerque	Albany	Monday	12:27 PM
Airline_4...	1605487	Albuquerque	Albany	Monday	01:33 PM

Figure 8-10 Results of running the SELECT for a three-character airline name

Left outer join

A left outer join returns all the rows that an inner join returns plus one row for each of the other rows in the first table that do not have a match in the second table.

In this case, you want to get a list of all airlines and any available flights they might have. You want to list the airline regardless of whether the airline lists any flights available. Your SQL statement looks like the following one:

```
SELECT AIRLNM, FLIGHT_NUMBER, DEPARTURE, ARRIVAL, DAY_OF_WEEK, DEPARTURE_TIME,
ARRIVAL_TIME, SEATS_AVAILABLE
FROM FLGHT400.AIRLINE A
LEFT OUTER JOIN FLGHT400.FLIGHTS B
ON A.AIRLIN = B.AIRLINES ;
```

The results of running this select against these tables are shown in Figure 8-11.

AIRLNM	FLIGHT_N...	DEPARTURE	ARRIVAL	DAY_OF_WEEK	DEPARTURE_TIME	ARRIVAL_TIME
Airline_5	6227502	St. Petersburg	Winnipeg	Saturday	08:36 AM	10:36 AM
Airline_5	6327503	St. Petersburg	Winnipeg	Saturday	09:38 AM	11:38 AM
Airline_5	6427504	St. Petersburg	Winnipeg	Saturday	10:41 AM	12:41 PM
Airline_5	6527505	St. Petersburg	Winnipeg	Saturday	12:27 PM	02:27 PM
Airline_5	6627506	St. Petersburg	Winnipeg	Saturday	01:33 PM	03:33 PM
Airline_5	6727507	St. Petersburg	Winnipeg	Saturday	03:01 PM	05:01 PM
Airline_5	6827508	St. Petersburg	Winnipeg	Saturday	05:02 PM	07:02 PM
Airline_5	6927509	St. Petersburg	Winnipeg	Saturday	07:00 PM	09:00 PM
Airline_5	7127510	St. Petersburg	Winnipeg	Sunday	07:33 AM	09:33 AM
Airline_5	7227511	St. Petersburg	Winnipeg	Sunday	08:36 AM	10:36 AM
Airline_5	7327512	St. Petersburg	Winnipeg	Sunday	09:38 AM	11:38 AM
Airline_5	7427513	St. Petersburg	Winnipeg	Sunday	10:41 AM	12:41 PM
Airline_5	7527514	St. Petersburg	Winnipeg	Sunday	12:27 PM	02:27 PM
Airline_5	7627515	St. Petersburg	Winnipeg	Sunday	01:33 PM	03:33 PM
Airline_5	7727516	St. Petersburg	Winnipeg	Sunday	03:01 PM	05:01 PM
Airline_5	7827517	St. Petersburg	Winnipeg	Sunday	05:02 PM	07:02 PM
Airline_5	7927518	St. Petersburg	Winnipeg	Sunday	07:00 PM	09:00 PM
Airline_7	--	-	-	-	-	-
Airline_2	1110963	Anchorage	Albany	Monday	08:01 AM	10:01 AM
Airline_2	1210964	Anchorage	Albany	Monday	09:03 AM	11:03 AM
Airline_2	1310965	Anchorage	Albany	Monday	10:06 AM	12:06 PM
Airline_2	1410966	Anchorage	Albany	Monday	11:09 AM	01:09 PM
Airline_2	1510967	Anchorage	Albany	Monday	12:55 PM	02:55 PM

Figure 8-11 Results when using the left outer join

Although Airline_7 has no flights that are available, it is still listed because the left outer join brings in all data from the first table even if no data exists in the second table.

Because no flight information exists for Airline_7, the fields from the FLIGHTS table returns NULLs to your program. Alternatively, you can create a view with the preceding SQL statement and **COALESCE** the NULLs into BLANKS. For example:

```
COALESCE(FLIGHT_NUMBER, ' ') as FLIGHT_NUMBER
```

Right outer join

A right outer join returns all the rows that an inner join returns plus one row for each of the other rows in the second table that do not have a match in the first table. It is the same as a left outer join with the tables specified in the opposite order.

For example, start by assuming that you have a Customer Master table (Figure 8-12 on page 265) and a Flights Order table (Figure 8-13 on page 265).

CUSTOMER_NO	CUSTOMER_NAME
30	Doe, Bob
493	Doe, Jane
298	Anderson, Bob
2440	Anderson, John
1785	Smith, Bob
5599	Smith, John
7774	Smith, Jane

Figure 8-12 Customer Master table

ORDER_NUMBER	AGENT_NO	CUSTOMER_NO	FLIGHT_NUMBER	DEPARTURE_DATE	TICKETS_ORDERED	CLASS	SEND_SIGNATURE_WI...
4969782	6	2976	6169516	2004-12-11 07:54:00.000000	13	N	N
4969783	3	1301	5185969	2004-03-26 07:33:00.000000	13	N	N
4969784	4	4617	2148898	2004-03-02 07:19:00.000000	13	N	N
4969785	4444	8367	5159194	2004-03-19 07:33:00.000000	13	N	N
4969786	5	5608	5195797	2004-10-22 07:26:00.000000	13	N	N
4969787	9	6242	4119306	2004-03-18 08:01:00.000000	13	N	N
4969788	2	3702	5102393	2004-06-04 07:05:00.000000	13	N	N
4969789	2	4668	2102429	2004-05-18 07:19:00.000000	13	N	N
4969790	9	8541	3108064	2004-02-18 07:47:00.000000	13	N	N
4969791	9	3324	5113670	2004-10-15 07:05:00.000000	13	N	N
4969792	4444	7712	1125705	2004-12-06 07:26:00.000000	13	N	N
4969793	8	1143	3102898	2004-02-25 07:54:00.000000	13	N	N
4969794	4	9504	5113185	2004-02-06 07:26:00.000000	13	N	N
4969795	2	5995	4114833	2004-10-21 07:12:00.000000	13	N	N
4969796	8	7087	5148232	2004-08-20 07:26:00.000000	13	N	N
4969797	8	6510	7113392	2004-06-27 07:26:00.000000	13	N	N
4969798	3	7050	1135533	2004-02-23 07:40:00.000000	13	N	N
4969799	5	4860	6125227	2004-06-26 07:33:00.000000	13	N	N
4969800	9	6782	4175943	2004-04-22 07:12:00.000000	13	N	N
4969801	9	1950	1107397	2004-02-09 07:40:00.000000	13	N	N
4969802	7	5276	4163091	2004-06-03 08:01:00.000000	13	N	N
4969803	2	3398	2117297	2004-11-16 07:12:00.000000	13	N	N
4969804	4	1652	5153461	2004-08-27 07:40:00.000000	13	N	N
4969805	3	3208	7102890	2004-06-27 07:26:00.000000	13	N	N

Figure 8-13 Flights Order table

In this situation, a right outer join might look like this:

```
SELECT ORDER_NUMBER, AGENT_NO, A.CUSTOMER_NO, FLIGHT_NUMBER, DEPARTURE_DATE,
TICKETS_ORDERED, CLASS, SEND_SIGNATURE, B.CUSTOMER_NO, CUSTOMER_NAME
FROM FLGHT400.ORDERS A
RIGHT OUTER JOIN FLGHT400.CUSTOMERS B
ON A.CUSTOMER_NO = B.CUSTOMER_NO
ORDER BY B.CUSTOMER_NO;
```

The results are similar to what is shown in Figure 8-14.

ORDER_N...	AGENT_NO	CUSTOMER_NO	FLIGHT_N...	DEPARTURE_DATE	TICKETS_ORDERED	CLASS	SEND_SIGNATURE_WI...	CUSTOMER_NO	CUSTOMER_NAME
5554872	9	2129	1118441	2004-08-09 07:47:00.000000	13	N	N	2129	Doe, Bob
5574480	4	2129	5187103	2004-09-24 07:05:00.000000	13	N	N	2129	Doe, Bob
5577354	3	2129	7104780	2004-10-10 07:12:00.000000	13	N	N	2129	Doe, Bob
5580285	2	2129	4109270	2004-05-27 07:47:00.000000	13	N	N	2129	Doe, Bob
5592643	8	2129	1130411	2004-07-19 07:19:00.000000	13	N	N	2129	Doe, Bob
5606588	4	2129	3163946	2004-04-14 07:12:00.000000	13	N	N	2129	Doe, Bob
5608963	4	2129	6107272	2004-07-10 08:01:00.000000	13	N	N	2129	Doe, Bob
5650736	9	2129	1147925	2004-11-15 07:19:00.000000	13	N	N	2129	Doe, Bob
-	-	-	-	-	-	-	-	2130	Anderson, John
4978491	6	2131	7194473	2004-07-11 07:26:00.000000	13	N	N	2131	Smith, Jane
4983348	3	2131	5118855	2004-10-08 07:26:00.000000	13	N	N	2131	Smith, Jane
4984657	3	2131	2154757	2004-04-06 07:19:00.000000	13	N	N	2131	Smith, Jane
5002399	4444	2131	6196858	2004-06-26 07:33:00.000000	13	N	N	2131	Smith, Jane
5011095	2	2131	7138044	2004-10-17 07:40:00.000000	13	N	N	2131	Smith, Jane
5016576	9	2131	3187337	2004-03-03 07:40:00.000000	13	N	N	2131	Smith, Jane
5046228	7	2131	4108199	2004-02-19 07:19:00.000000	13	N	N	2131	Smith, Jane
5053318	9	2131	6116363	2004-02-21 08:01:00.000000	13	N	N	2131	Smith, Jane
5053876	7	2131	7112806	2004-11-21 07:40:00.000000	13	N	N	2131	Smith, Jane
5067832	8	2131	1165017	2004-10-18 07:19:00.000000	13	N	N	2131	Smith, Jane
5068449	6	2131	1165017	2004-10-18 07:19:00.000000	13	N	N	2131	Smith, Jane
5111544	2	2131	4106227	2004-10-14 07:12:00.000000	13	N	N	2131	Smith, Jane
5115713	7	2131	7178406	2004-06-15 07:40:00.000000	13	N	N	2131	Smith, Jane
5121410	5	2131	5164738	2004-04-23 07:33:00.000000	13	N	N	2131	Smith, Jane

Figure 8-14 Results for the right outer join

Although customer number 2130 has not purchased any flights, they are still listed because the right outer join brings in all data from the CUSTOMER table (second) even if no data exists in the ORDERS table (first).

Because no ORDER information exists for customer number 2130, the fields from the ORDERS table return NULLs to your program. Alternatively, you can create a view with the above SQL statement and **COALESCE** the NULLs into BLANKS. For example:

```
COALESCE(ORDER_NUMBER, ' ') as ORDER_NUMBER
```

Left exception join

A left exception join returns only the rows from the first table that do not have a match in the second table.

For example, suppose that you need to know what airlines have no flights that are scheduled. Your SQL statement looks like the following one:

```
SELECT AIRLNM, FLIGHT_NUMBER, DEPARTURE, ARRIVAL, DAY_OF_WEEK, DEPARTURE_TIME,
ARRIVAL_TIME, SEATS_AVAILABLE
FROM FLGHT400.AIRLINE A
LEFT EXCEPTION JOIN FLGHT400.FLIGHTS B
ON A.AIRLIN = B.AIRLINES;
```

The results are similar to what is shown in Figure 8-15.

AIRLNM	FLIGHT_N...	DEPARTURE	ARRIVAL	DAY_
Airline_7	--	-	-	-

Figure 8-15 Results of the left exception join

In Figure 8-15, Airline_7 currently has no flights that are scheduled. Because no flight information exists for Airline_7, the fields from the FLIGHTS table return NULLs to your program. Alternatively, you can create a view with the above SQL statement and **COALESCE** the NULLs into BLANKS. For example:

```
COALESCE(FLIGHT_NUMBER, ' ') as FLIGHT_NUMBER
```

Right exception join

A right exception join returns only the rows from the second table that do not have a match in the first table.

For example, assume that you have a Customer Master table as shown in Figure 8-12 on page 265 and a Flights Order Table as shown in Figure 8-13 on page 265. In this situation, a right outer join might look like the following code:

```
SELECT ORDER_NUMBER, AGENT_NO, A.CUSTOMER_NO, FLIGHT_NUMBER, DEPARTURE_DATE,
TICKETS_ORDERED, CLASS, SEND_SIGNATURE, B.CUSTOMER_NO, CUSTOMER_NAME
FROM FLGHT400.ORDERS A
RIGHT EXCEPTION JOIN FLGHT400.CUSTOMERS B
ON A.CUSTOMER_NO = B.CUSTOMER_NO
ORDER BY B.CUSTOMER_NO;
```

The results are similar to what is shown in Figure 8-16 on page 267.

DEPARTURE_DATE	TICKETS_ORDERED	CLASS	SEND_SIGNATURE...	CUSTOMER_NO	CUSTOMER_NAME
--	--	--	-	2130	Smith, Jane
--	--	--	-	2205	Anderson, John
--	--	--	-	7528	Doe, Jane
--	--	--	-	7814	Smith, John
--	--	--	-	8002	Doe, Bob
--	--	--	-	8227	Anderson, Bob
--	--	--	-	9862	Smith, Bob

Figure 8-16 Results for the right exception join

Because no ORDER information exists for these customers, the fields from the ORDERS table return NULLs to your program. Alternatively, you can create a view with the above SQL statement and **COALESCE** the NULLs into BLANKS. For example:

```
COALESCE(ORDER_NUMBER, ' ') as ORDER_NUMBER
```

Full outer join

Like the left and right outer joins, a full outer join returns matching rows from both tables. However, a full outer join also returns nonmatching rows from both tables.

Cross join

A cross join, also known as a Cartesian product join, returns a result table where each row from the first table is combined with each row from the second table. Cross joins can return a large result set. For example, if you have table_a with 10000 rows and table_b with 150000 rows, a cross join returns 1,500,000,000 rows because every row in table_a is joined with every row in table_b.

8.3.8 Commitment control

Commitment control is a powerful feature that helps ensure the integrity of a database. You can use it to ensure that a set of related changes across one or more tables in a database are not permanently applied until a program commits them. Alternatively, a program might roll back any of the changes instead of committing them. Many people consider commitment control a necessity in the maintenance of a database.

Today, things have changed to make the usage of commitment control more acceptable:

- ▶ Systems are now bigger and faster, and there is much less resistance to the usage of journals.
- ▶ RPG IV allows optional usage of commitment control in programs.
- ▶ ILE provides a means of limiting the scope of commitment control within a job.
- ▶ The IBM i journal support continues to improve.

Transaction

A transaction is a group of individual changes to objects on the system that appear as a single atomic change to the user.

A transaction can be any of the following situations:

- ▶ Inquiries in which no database file changes occur.
- ▶ Simple transactions that change one database file.
- ▶ Complex transactions that change one or more database files.

- ▶ Complex transactions that change one or more database files, but these changes represent only a part of a logical group of transactions.
- ▶ Simple or complex transactions that involve database files at more than one location. The database files can be in one of the following situations:
 - On a single remote system.
 - On the local system and one or more remote systems.
 - Assigned to more than one journal on the local system. Each journal can be thought of as a local location.
- ▶ Simple or complex transactions on the local system that involve objects other than database files.

IBM i Navigator uses the term transaction, and the character-based interface uses the term logical unit of work (LUW). The two terms are interchangeable. This topic, unless specifically referring to the character-based interface, uses the term transaction. Figure 8-17 illustrates the commit control flow.

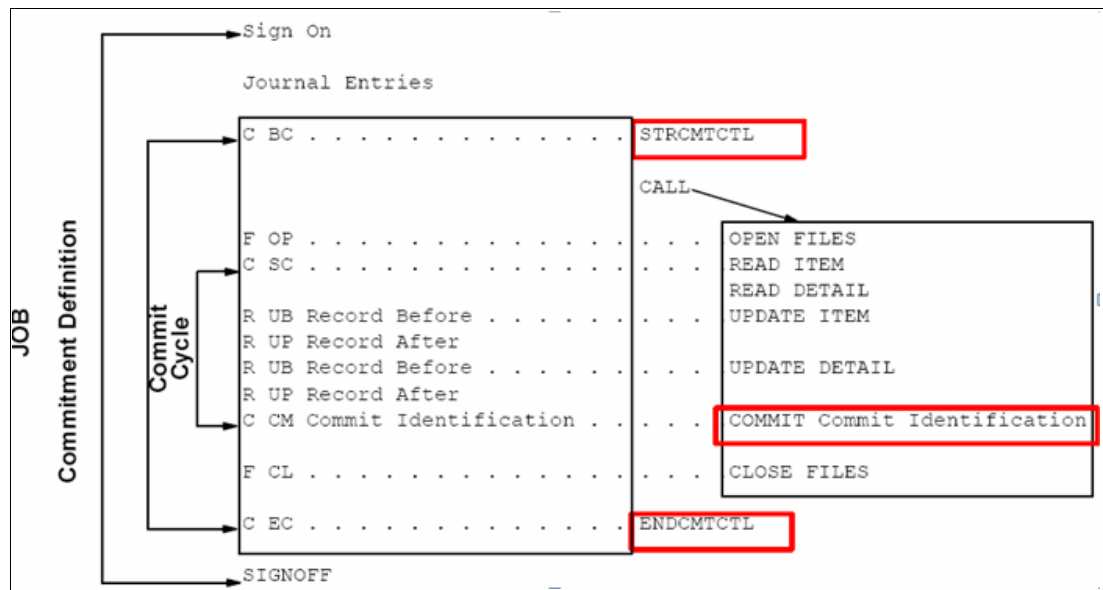


Figure 8-17 Commitment control flow illustration

When a program is processing a transaction that consists of writing, updating, and deleting one or more rows across multiple tables in a database, commitment control offers a means of treating the transaction as a single unit. For example, commitment control provides a way for a program to identify and process an order as a single transaction even though that order is spread over many rows on many tables, and adding the order involves updating columns in many rows on other tables (for example, stock figures and customer balance information).

Imagine what would happen if your system suddenly lost power while a program (or a number of programs) was processing a transaction. When the system performs an initial program load (IPL), the databases contain many incomplete transactions. However, if the programs were using commitment control, the system automatically rolls back any uncommitted rows from the databases.

Now, imagine what would happen if a program that is processing a transaction fails because of a bug. There are incomplete transactions in the database. If the program uses commitment control, you can roll back the incomplete transactions.

Therefore, commitment control should be a consideration for any transaction that consists of more than one row on one or more tables.

How commitment control works

Commitment control ensures that either the entire group of individual changes occurs on all systems that participate or that none of the changes occur.

Commitment control allows you to complete the following tasks:

- ▶ Ensure that all changes within a transaction are completed for all resources that are affected.
- ▶ Ensure that all changes within a transaction are removed if processing is interrupted.
- ▶ Remove changes that are made during a transaction when the application determines that a transaction is in error.

You can also design an application so that commitment control can restart the application if a job, an activation group within a job, or the system ends abnormally. With commitment control, you can have assurance that when the application starts again, no partial updates are in the database because of incomplete transactions from a prior failure.

Commitment control scoping

ILE introduces two changes for commitment control:

- ▶ Multiple, independent commitment definitions per job. Transactions can be committed and rolled back independently of each other. Before ILE, only a single commitment definition was allowed per job.
- ▶ If changes are pending when an activation group ends normally, the system implicitly commits the changes. Before ILE, the system did not commit the changes.

Commitment control allows you to define and process changes to resources, such as database files or tables, as a single transaction. A transaction is a group of individual changes to objects on the system that should appear to the user as a single atomic change.

Commitment control ensures that one of the following occurs on the system:

- ▶ The entire group of individual changes occurs (a commit operation)
- ▶ None of the individual changes occur (a rollback operation)

Various resources can be changed under commitment control by using both OPM programs and ILE programs.

The Start Commitment Control (**STRCMTCTL**) command, which is shown in Figure 8-18, makes it possible for programs that run within a job to make changes under commitment control. When commitment control is started by using the **STRCMTCTL** command, the system creates a commitment definition. Each commitment definition is known only to the job that issued the **STRCMTCTL** command. The commitment definition contains information pertaining to the resources that are being changed under commitment control within that job. The commitment control information in the commitment definition is maintained by the system as the commitment resources change. The commitment definition is ended by using the End Commitment Control (**ENDCMTCTL**) command.

```

Start Commitment Control (STRCMTCTL)

Type choices, press Enter.

Lock level . . . . . LCKLVL          _____
Notify object:      NFYOBJ
  Object . . . . .                *NONE_____
  Library . . . . .                _____
  Object type:
  (*MSGQ *DTAARA or *FILE) . . . . _____
  Member, if *FILE . . . . .       _____
Commitment definition scope . . CMTSCOPE *ACTGRP
Text 'description' . . . . . TEXT    *DFTTEXT
-----
Journal . . . . . DFTJRN            *NONE_____
  Library . . . . .                _____
Journal entries to be omitted . OMTJRNE *NONE_____

Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys

```

Figure 8-18 STRCMTCTL - Start Commitment Control CL command

Commitment definitions and activation groups

Multiple commitment definitions can be started and used by programs that are running within a job. Each commitment definition for a job identifies a separate transaction that has resources that are associated with it. These resources can be committed or rolled back independently of all other commitment definitions started for the job.

Note: Only ILE programs can start commitment control for activation groups other than a default activation group. Therefore, a job can use multiple commitment definitions only if the job is running one or more ILE programs.

Original program model (OPM) programs run in the single-level storage default activation group. By default, OPM programs use the ***DFACTGRP** commitment definition. For OPM programs, you can use the ***JOB** commitment definition by specifying **CMTSCOPE(*JOB)** on the **STRCMTCTL** command.

When you use the Start Commitment Control (**STRCMTCTL**) command, you specify the scope for a commitment definition on the commitment scope (**CMTSCOPE**) parameter. The scope for a commitment definition indicates which programs that run within the job use that commitment definition. The default scope for a commitment definition is to the activation group of the program issuing the **STRCMTCTL** command. Only programs that run within that activation group, as shown in Figure 8-19, use that commitment definition, except that the default activation groups share one commitment definition. Commitment definitions that are scoped to an activation group are referred to as commitment definitions at the activation-group level.

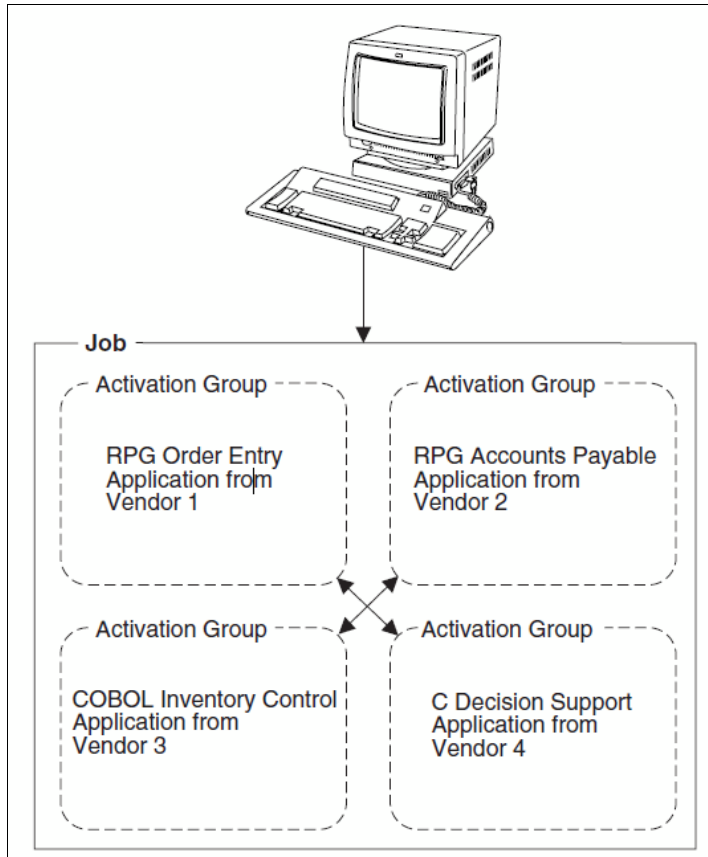


Figure 8-19 Commit control within an activation group

The commitment definition that starts at the activation-group level for a default activation group is known as the default activation-group (***DFACTGRP**) commitment definition. Commitment definitions for many activation-group levels can be started and used by programs that run within various activation groups for a job.

A commitment definition can also be scoped to the job. A commitment definition with this scope value is referred to as the job-level or ***JOB** commitment definition. Any program that is running in an activation group that does not have a commitment definition that is started at the activation-group level uses the job-level commitment definition. This occurs if the job-level commitment definition has already been started by another program for the job. Only a single job-level commitment definition can be started for a job.

For an activation group, only a single commitment definition can be used by the programs that run within that activation group. Programs that run within an activation group can use the commitment definition at either the job level or the activation-group level. However, they cannot use both commitment definitions at the same time.

When a program performs a commitment control operation, the program does not directly indicate which commitment definition to use for the request. Instead, the system determines which commitment definition to use based on which activation group the requesting program is running in. This is possible because, at any point in time, the programs that run within an activation group can use only a single commitment definition.

Ending commitment control

Commitment control can be ended for either the job-level or activation-group-level commitment definition by using the End Commitment Control (**ENDCMTCTL**) command. The **ENDCMTCTL** command indicates to the system that the commitment definition for the activation group of the program that is making the request should be ended. The **ENDCMTCTL** command ends one commitment definition for the job. All other commitment definitions for the job remain unchanged.

If the commitment definition at the activation-group level is ended, programs that are running within that activation group can no longer make changes under commitment control. If the job-level commitment definition is started or exists, any new file open operations specifying commitment control use the job-level commitment definition.

If the job-level commitment definition is ended, any program that is running within the job that was using the job-level commitment definition can no longer make changes under commitment control. If commitment control is started again with the **STRCMTCTL** command, changes can be made.

Implementing commitment control

In this section, you can see how to implement commitment control. It is a simple procedure and you can do it in any application.

Journals and journal receivers

Commitment control requires that the tables that are involved in the transaction that is being committed are all attached to the same journal.

The basic concept behind journals is to offer a means of database recovery up to a specific point. When a table is attached to a journal, the database manager records a copy of every row in the table that is added updated or deleted. This means that the system has a copy of every change that was made to an attached table so that, in the case of data loss, the system has a means of recovering all changes that were made to that table.

The journal process consists of two parts: a journal and a journal receiver. Journals are created with the Create Journal (**CRTJRN**) command and journal receivers are created with the Create Journal Receiver (**CRTJRNRCV**) command. When a journal is created, it is attached to a journal receiver. You can specify that changes made to a table are recorded in a journal by using the Start Journal Physical File (**STRJRNPF**) command. You can choose to record before images, after images, or both before and after images. Figure 8-20 on page 273 gives an illustration of how this process works.

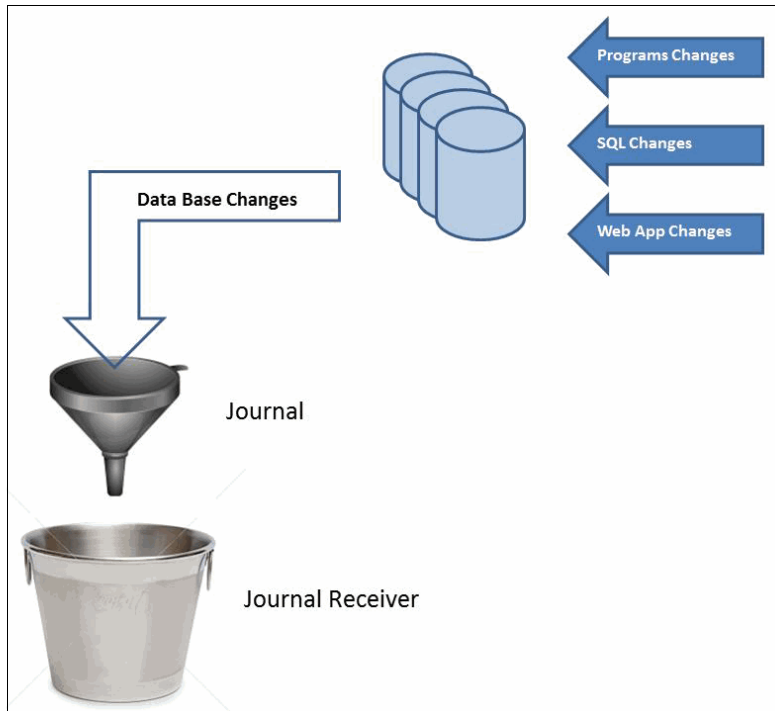


Figure 8-20 The journal process

The journal is depicted as a funnel and a bucket represents the journal receiver. As changes are made to tables, a copy of the changes is sent to the journal, which then drops the copies into the attached journal receiver. When a journal receiver is full, a new journal receiver can be created and attached to the journal in place of the original. This is a simple means of managing the amount of space that is occupied by the journal receivers.

If your tables are already attached to journals, there is nothing else you need to do to implement commitment control, apart from the applying the required code in your programs.

If you are using journals for database recovery (or high availability), you more than likely have a strategy where all tables (regardless of application) are attached to a single journal. If you do not require journals for database recovery, you might want to consider having separate journals per application or per schema (library), as is the default when you create a schema with SQL.

Creating a test schema

Example 8-19 shows the SQL that is used to create a schema that is named COMMIT, which contains two related tables, called HEADER and DETAILS.

Example 8-19 Example of creating a schema

```
(A) CREATE SCHEMA "COMMIT" ;

(B) CREATE TABLE COMMIT/HEADER (
    "KEY" CHAR(2) CCSID 37 NOT NULL DEFAULT '' ,
    TEXT CHAR(20) CCSID 37 NOT NULL DEFAULT '' ,
    NUMROWS DECIMAL(5, 0) NOT NULL DEFAULT 0 ,
    CONSTRAINT COMMIT/HEADER_PRIMARY PRIMARY KEY( "KEY" ) ) ;

LABEL ON TABLE COMMIT/HEADER
IS 'Header Rows' ;
```

```

LABEL ON COLUMN COMMIT/HEADER
( "KEY" TEXT IS 'Key' ,
TEXT TEXT IS 'Header Text' ,
NUMROWS TEXT IS 'Number of Rows' ) ;

```

```

(C) CREATE TABLE COMMIT/DETAILS (
"KEY" CHAR(2) CCSID 37 NOT NULL DEFAULT ' ' ,
"SEQUENCE" DECIMAL(5, 0) NOT NULL DEFAULT 0 ,
TEXT CHAR(20) CCSID 37 NOT NULL DEFAULT ' ' ,
CONSTRAINT COMMIT/DETAIL_PRIMARY PRIMARY KEY( "KEY" ,
"SEQUENCE" ) ) ;

```

```

LABEL ON TABLE COMMIT/DETAILS
IS 'Detail Rows' ;

```

```

LABEL ON COLUMN COMMIT/DETAILS
( "KEY" TEXT IS 'Key' ,
"SEQUENCE" TEXT IS 'Key Seq.' ,
TEXT TEXT IS 'Detail Text' ) ;

```

```

(D) CREATE UNIQUE INDEX COMMIT/HEADER1
ON COMMIT/HEADER ( "KEY" ASC ) ;

```

```

COMMENT ON INDEX COMMIT/HEADER1
IS 'Header by Key' ;

```

```

CREATE UNIQUE INDEX COMMIT/DETAILS1
ON COMMIT/DETAILS ( "KEY" ASC , "SEQUENCE" ASC ) ;

```

```

COMMENT ON INDEX COMMIT/DETAILS1
IS 'Details by Key' ;

```

```

(E) ALTER TABLE COMMIT/DETAILS
ADD CONSTRAINT COMMIT/DETAIL_TO_HEADER
FOREIGN KEY( "KEY" )
REFERENCES COMMIT/HEADER ( "KEY" )
ON DELETE CASCADE
ON UPDATE RESTRICT ;

```

Here are the main points to note (refer to the corresponding letters in Example 8-19 on page 273):

- ▶ A: Creating the schema COMMIT results in a library that is named COMMIT that contains a journal that is named QSQJRN and a corresponding journal receiver named QSQJRN0001. All tables that are created in the schema are automatically attached to the journal QSQJRN.
- ▶ B: The table HEADER consists of a key, a text field, and a numeric field containing the number of corresponding DETAIL rows for the key. HEADER has a primary key constraint that is based on the key field.
- ▶ C: The table DETAILS consists of a key (which corresponds to the key on the header table), a sequence number, and a text field. DETAILS has a primary key constraint that is based on the key field and the sequence number.

- ▶ D: An index is created for each table (HEADER1 and DETAILS1). The index keys correspond to the definition of the primary key constraint for each table. RPG programs use these indexes to access the rows in the tables.
- ▶ E: A foreign key constraint is defined between the DETAILS and HEADER tables. The constraint contains a cascade delete rule, which means that when a row is deleted from the HEADER table all corresponding rows (based on the key fields) are automatically deleted from the DETAILS table. A cascade delete is only possible when the files are attached to the same journal.

Because this is a schema, the HEADER and DETAILS tables are automatically attached to the journal QSQJRN. If you intend to implement commitment control on an existing database, you must ensure that each table (or physical file) is attached to the same journal.

Having the journal in the same schema as the tables is not a requirement for commitment control; the only requirement for commitment control is that the tables are attached to the same journal.

Example 8-20 shows an RPG program that looks at populating a database and testing commit control.

Example 8-20 Example of an ILE RPG IV program populating the database and testing commit control

```

Ctl-opt Option(*SrcStmt : *NoDebugIO)
          DftActGrp(*No) ActGrp('COMMITDEMO');

(A)  Dcl-f Header1 Usage(*UPDATE : *OUTPUT)
          Keyed
          Commit(SomeTimes);
      Dcl-f Details1 Usage(*UPDATE:*OUTPUT)
          Keyed
          Commit(SomeTimes);

      Dcl-pr Commit1 ExtPgm('COMMITRPG1');
          SomeTimes Ind;
      End-pr;

(B)  Dcl-pi Commit1;
          SomeTimes Ind;
      End-pi;

      Dcl-s ToDo Char(1);

(C)  Dsply 'Enter a Key Value (2 long): ' ' ' Key;

(D)  If (Key <> *Blanks);
          Text = 'Header for ' + Key;
          Write Header;
          For Sequence = 1 to 3;
              Text = 'Detail for ' + Key + ' ' +
                    %Char(Sequence);
              Write Details;
          EndFor;
(E)  Chain Key Header1;
          NumRows += 1;
          Update Header;
      EndFor;
      EndIf;

```

```

      ToDo = *Blanks;
(F)  Dow (ToDo <> 'c' and ToDo <> 'r' and ToDo <> 'i');
      Dsply 'c - Commit, r - Rollback, i - Ignore '
          ' ' ToDo;
      EndDo;

(G)  If SomeTimes and (ToDo = 'c');
      Commit;
(H)  ElseIf SomeTimes and (ToDo = 'r');
      RolBk;
      EndIf;
(I)

      *InLR = *On;

```

This program prompts for a key, writes one record to the HEADER table and three records to the DETAILS table for the key value entered. Every time a row is added to the DETAILS table, the Number of Rows on the corresponding HEADER row is incremented. The program then prompts to commit, roll back, or ignore the transaction. Here are the main points to note (refer to the corresponding numbers in Example 8-20 on page 275):

- ▶ A: The program uses the indexes HEADER1 and DETAILS1 to access the database. Both tables specify the **COMMIT** keyword to indicate that commitment control can be used on the tables. The **SomeTimes** indicator controls whether (*ON) or not (*OFF) commitment control is active when the files are opened.
- ▶ B: The indicator **SomeTimes** is passed as a parameter when the program is called.
- ▶ C: The program requests the entry of a key value.
- ▶ D: If a key value is entered, the program adds a record to the header table and three corresponding records to the details table.
- ▶ E: The Number of Rows on the HEADER rows is incremented for each row added to DETAILS.
- ▶ F: The program prompts for an entry to determine what should be done with the rows just added to the tables.
- ▶ G: If the entry was “c” for commit, then the rows are committed. The **SomeTimes** indicator also conditions the commit operation. Issuing a commit or rollback operation when commitment control is not active results in runtime error RNQ0802:
 COMMIT or ROLBK operation attempted when commitment control was not active (C G D F).
- ▶ H: If the entry was “r” for rollback, then the rows that were just written are removed from the tables. As with the commit operation, the rollback operation is also conditioned by the **SomeTimes** indicator.
- ▶ I: No action takes place if the entry was “i” for “ignore.” Tables that opened while commitment control is enabled must be attached to the same journal. If either of the tables are not attached to a journal, or if the tables are not attached to the same journal, the system issues the following message:

```
Member *N not journaled to journal *N.
```

Normal program call with no commit

The following example shows how to call the program without commitment control (the indicator parameter must be in single quotation marks):

```
CALL PGM(COMMITRPG1) PARM('0')
```

When prompted, enter a value of 'aa' for the key. It is irrelevant what value you enter for the commitment option because the program is not performing commitment control. By using the **run query** command to look at the contents of the two tables, you see that they have the values that are shown in Example 8-21. A single row was added to the HEADER table and three corresponding rows were added to the DETAILS table.

Example 8-21 Result of calling the program with no commit

KEY	TEXT	NUMROWS
aa	Header for aa	3

KEY	SEQUENCE	TEXT
aa	1	Detail for aa 1
aa	2	Detail for aa 2
aa	3	Detail for aa 3

Program call with commit

Now, call the program with commitment control enabled by running the following command:

```
CALL PGM(COMMITRPG1) PARM('1')
```

The program fails with the following message:

```
Commitment definition *N not valid for open of DETAILS1
```

The second-level message text for the error provides more information, but the issue is that you did not specify that you are using commitment control in the job. It is not enough to simply have the **COMMIT** keyword in the file specifications in the RPG program. You must also specify that you are using commitment control in the job. Run the Start Commitment Control (**STRCMTCTL**) command as follows:

```
STRCMTCTL LCKLVL(*CHG) CMTSCOPE(*JOB)
          TEXT('Test Commitment Control')
```

The command specifies a lock level of ***CHG** and a commit scope of ***JOB**. ***JOB** is not the default value for the commit scope parameter; the default value is ***ACTGRP** for activation group.

Test commit

Now, look at your three main options when using commitment control (specify the parameter value '1' on all calls to the program):

- Call the program and enter a value of "bb" for the key and a value of "c" for the commitment option. The tables should now contain the values that are shown in Example 8-22. An additional row was added to the HEADER table and three corresponding rows were added to the DETAILS table.

Example 8-22 Result of calling with commit enabled

KEY	TEXT	NUMROWS
aa	Header for aa	3
bb	Header for bb	3

KEY	SEQUENCE	TEXT
aa	1	Detail for aa 1
aa	2	Detail for aa 2
aa	3	Detail for aa 3
bb	1	Detail for bb 1

```
bb      2  Detail for bb 2
bb      3  Detail for bb 3
```

- ▶ Call the program and enter a value of “cc” for the key and a value of “r” (roll back) for the commitment option. When you look at the contents of the two tables, you see that they remain unchanged. The new rows were not added and the contents are as they were in Example 8-22 on page 277. The ROLBK operation in the program removed the four rows that were added to the two tables.
- ▶ Call the program and enter a value of “dd” for the key and a value of “i” (ignore) for the commitment option. What do you expect to find on the tables? Will the new rows appear in the table or do they not appear? At first glance, it appears that the new rows are on the tables, as shown in Example 8-23. Even if you signed into another job and viewed the contents of the tables, the new rows appear to be there.

Example 8-23 Result of rows being added but not yet committed

```
KEY TEXT                NUMROWS
aa  Header for aa      3
bb  Header for bb      3
dd  Header for dd      3
```

```
KEY SEQUENCE TEXT
aa      1  Detail for aa 1
aa      2  Detail for aa 2
aa      3  Detail for aa 3
bb      1  Detail for bb 1
bb      2  Detail for bb 2
bb      3  Detail for bb 3
dd      1  Detail for dd 1
dd      2  Detail for dd 2
dd      3  Detail for dd 3
```

However, this is not the full story. Although the new rows appear in the tables, they are only available to other programs if the tables are open for input only. Use the Display Record Locks (**DSPRCDLCK**) command to see that the three newly inserted rows are all locked, as shown in Example 8-24. These rows are not available for update to any other programs. Although the rows are physically added to the tables, they have not yet had a commit or a rollback instruction issued to them.

Example 8-24 Result of DSPRCDLCK

```
Display Member Record Locks

File . . . . . : DETAILS      Member . . . . . : DETAILS
Library . . . . . : COMMIT

                                System: xxxxxxxx

Record
Number Job      User      Number Status Lock
Type
      7 COMCONPTA TUOHYP   115363 HELD  UPDATE
      8 COMCONPTA TUOHYP   115363 HELD  UPDATE
      9 COMCONPTA TUOHYP   115363 HELD  UPDATE
```

Run the **ROLLBACK** command. The pending changes are removed and do not appear if you view the contents of the tables. Alternatively, you can enter the **COMMIT** command to have the pending changes applied.

Ending commitment control

At any stage in a job, you can end commitment control by running the End Commitment Control (**ENDCMTCTL**) command. There are no parameters for this command.

If there are pending changes when you end commitment control, you receive the following message:

```
ENDCMTCTL requested with changes pending.  
(RB C CM)
```

Entering the value RB (the default reply) indicates that a rollback should be performed. Entering the value CM indicates that a commit should be performed. Entering the value C indicates that the End Commitment Control command should be canceled.

If there are pending changes when you end a job, an implicit rollback is performed before the job ends.

8.4 Database modularization

Here are some of the common goals throughout this publication:

- ▶ Separate your program from the underlying data model.
- ▶ Separate the User Interface (UI View) from the business layer (Controller) from the Database (Model).
- ▶ Modularize not just your program but the entire application.
- ▶ Design your systems to be expandable to other languages (PHP, Java, .NET, and Ruby).

Stored procedures, triggers, and user-defined functions (UDF) offer a powerful way of architecting an application to achieve these goals. These items are database artifacts that the database management system (DBMS) can run on your behalf whether you call them explicitly or the DBMS runs them automatically because of an event within the DBMS.

8.4.1 Stored procedures

With the introduction of IBM i 7.1, RPG programs can now consume result sets that are produced by stored procedures. It is now possible to have one stored procedure that returns data in a result set that all languages can read, as shown in the architectural design that is shown in Figure 8-21.

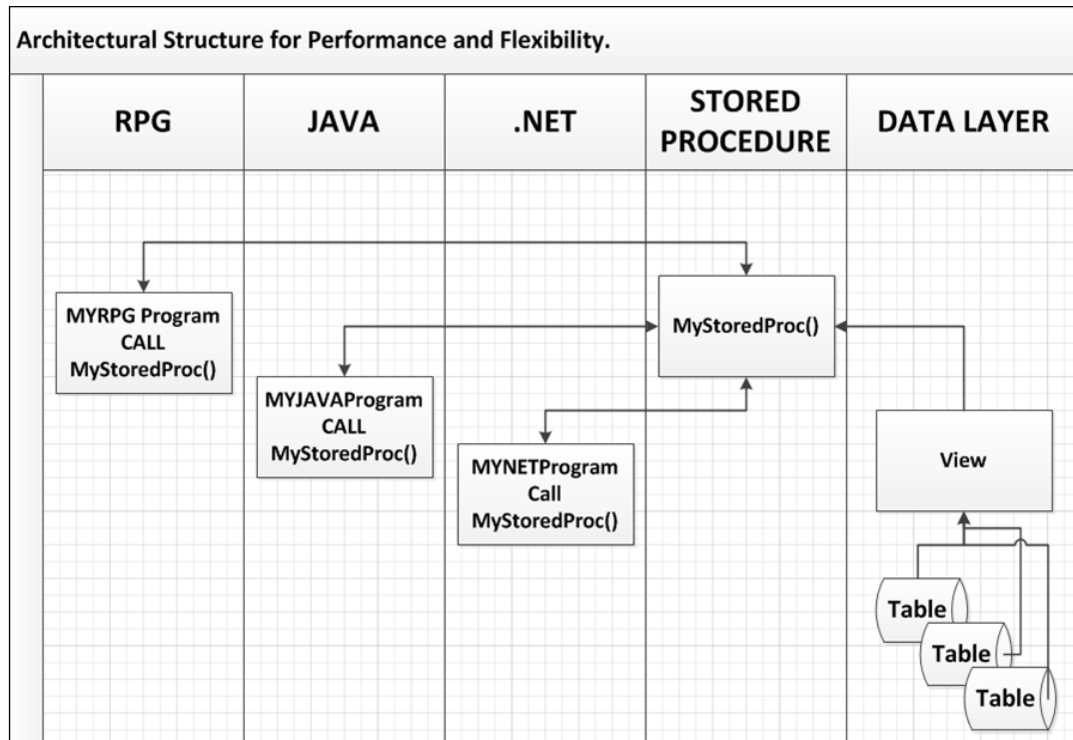


Figure 8-21 Architectural structure for performance and flexibility

This design is a powerful architectural design because of the following reasons:

- ▶ All programs, regardless of the language, have a standard way of calling a process to retrieve data.
- ▶ All programs receive the same answer. If there is an issue that must be corrected, it is in one place.
- ▶ Changes to your data access module that are needed (for example, the stored procedure that is described above) are done in one spot and not in all the various programs.
- ▶ Fewer programs are accessing the physical tables, making it less likely that database changes cause an issue.
- ▶ The industry-standard architecture is compatible with various DBMSs.

To use a result set in your RPG programs, a new SQL data type must be defined. The SQL type is `RESULT_SET_LOCATOR`.

For more information, see *Stored Procedures, Triggers, and User-Defined Functions on DB2 Universal Database for iSeries*, SG24-6503.

Example of creating and using a stored procedure

Using our FLIGHT400 company, a business analyst needs a report of all customers that tells the analyst what flight the customers are on with additional information about the flight (class, ticket number, departure time, and so on). To run this report, complete the following steps:

1. Create a view that brings all this information together into one set of rows. This is in contrast to the traditional way an RPG program is coded by either accessing each physical file (PF) separately or creating a joined logical file that the system constantly must maintain. In this example, we call our view CUSTORDV1 and it contains the following fields:

```
ORDRNO,AGNTNO,CUSTNM,CUSTNO,FLGTNO,DEPART,TCKTNO, CLASS,SENDSG
```

2. Create a stored procedure that is named GET_CUSTOMER_ORDERS. It returns a result set that is defined by the view. Basically, it returns an open cursor with the following statement:

```
SELECT ORDRNO,AGNTNO,CUSTNM, CUSTNO,FLGTNO,DEPART,TCKTNO, CLASS,SENDSG FROM CUSTORDV1.
```

3. Create a field of type RESULT_SET_LOCATOR that is defined in the RPG program:

```
DrsCustOrders      S          SQLTYPE(RESULT_SET_LOCATOR)
```

In this example, rsCustOrders is a variable of type RESULT_SET_LOCATOR that is used to consume the rows that are retrieved from the CUSTORDV1 view.

4. Define a data structure. The data structure is defined externally by the view and used to define a multiple occurrence data structure that holds rows that are read from the result set.

```
d CustOrderDS e ds          extname(CUSTORDV1)
d CustOrdersDS ds          l ikeds(CustOrderDS)
d                          occurs(10000)
```

5. Call the stored procedure:

```
Exec SQL
      CALL GET_CUSTOMER_ORDERS();
```

6. The result set that is created by the stored procedure is now in memory. Associate the RESULT_SET_LOCATOR with the result set that is returned by the stored procedure:

```
Exec SQL
      ASSOCIATE RESULT SET LOCATORS
      (:rsCustOrders)
      WITH PROCEDURE GET_CUSTOMER_ORDERS;
```

Note: If the stored procedure returns multiple result sets, you can associate multiple RESULT_SET_LOCATORS to each result set by separating the RESULT_SET_LOCATORS by commas. For example:

```
(:rsCustOrders, :rsCustomers, :rsOrders)
```

7. Create a cursor for the RESULT_SET_LOCATOR and open the cursor:

```
Exec SQL
      ALLOCATE CustomerOrders CURSOR
      FOR RESULT SET :rsCustOrders;
```

```
Exec SQL
      Open CustomerOrders;
```

Note: It is not necessary to explicitly open the cursor. However, it is preferred practice for readability.

8. Fetch from the cursor 10000 rows at a time (defined by the occurs). Get the number of rows read and loop through the multiple occurrence data structure performing whatever actions you need. At the end of the loop, read the next 10000 rows until you run out of records in the result set.

```
sqlcode      = *zeros;
LastRow = *Zeros;

//-----
//Loop until no more records to process
dow sqlcode = 0 and
    LastRow = *zeros;

exec sql
    fetch next
    from CustomerOrders
    for 10000 rows
    into :CustOrdersDS;

//-----
//Get # of rows fetched and last row indicator
exec sql
    get diagnostics :rowcount = ROW_COUNT,
                   :lastrow = DB2_LAST_ROW;

for CurrentRecord = 1 to rowcount ;
    %occur(CustOrdersDS) = CurrentRecord;
... do something ...

next;
enddo;
exec sql
    close CustomerOrders;
```

Now you have a program that uses a stored procedure. The stored procedure can be used by multiple different applications and languages. The stored procedure is separate from the database table so that any changes to the table that the stored procedure does not need does not affect the stored procedure. Here are some benefits of using stored procedures:

- ▶ You gain performance because the DBMS uses the stored procedure and the view, which helps your program receive data faster.
- ▶ You gain flexibility by separating your programs from the underlying data model.
- ▶ You gain compatibility by using industry standard architecture.

8.4.2 Triggers

Triggers are a powerful feature of DB2 for i. Triggers are application independent. They are user-written programs that are activated by the database manager when specific events occur within the database. Because triggers are application independent and run within the database management system (DBMS), they cannot be circumvented. No matter what application you run, if the event occurs that the DBMS is watching for, the trigger is activated.

Here are some uses for triggers:

- ▶ Enforcing business rules
- ▶ Tracking changes to specific tables in an audit log
- ▶ Encrypting columns as the column is updated or new rows are inserted
- ▶ Replicating data to Operational Data Store (ODS)
- ▶ Automatically updating a column in table_b when table_a is updated
- ▶ Making read-only SQL views updatable

You can benefit from triggers for several reasons:

- ▶ Application independence:
IBM DB2 Universal Database™ for iSeries activates the trigger program, regardless of the interface you are using to access the data. Rules that are implemented by triggers are enforced consistently by the system instead of by a single application.
- ▶ Easy maintenance
If you must change the business rules in your database environment, you must update or rewrite the triggers. No change is needed to the applications (they transparently comply with the new rules).
- ▶ Code reusability
Functions that are implemented at the database level are automatically available to all applications using that database. You do not need to replicate those functions throughout the different applications.
- ▶ Easier client/server application development
Client/server applications take advantage of triggers. In a client/server environment, triggers might provide a way to split the application logic between the client and the server system.

There are two types of triggers:

- ▶ SQL triggers
- ▶ External triggers

Triggers require exclusive lock on the tables as a trigger is being added or removed from the table itself. One flexible architecture for triggers is to create a SQL trigger that calls a stored procedure to do the processing. That way, if any business rules or other changes are required, you end up changing the stored procedure. You do not have to worry about getting exclusive lock on the table to make modifications. All the trigger does is call a processing storage procedure. This is a good option for shops that operate 24x7.

Figure 8-22 illustrates calling a store procedure from a trigger.

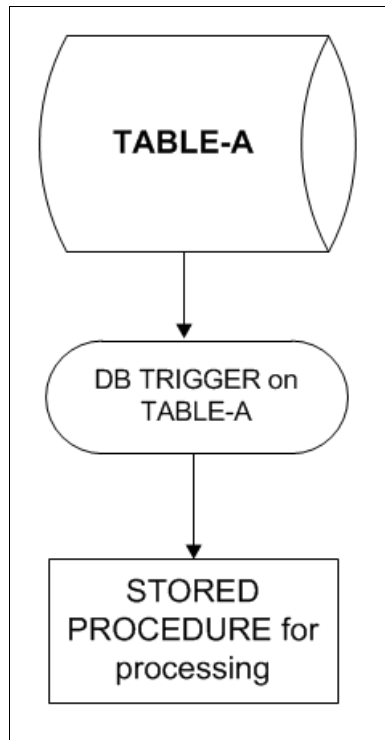


Figure 8-22 Illustration of calling a stored procedure from a trigger

In addition to standard before and after base table triggers, DB2 for i provides an *Instead of Trigger (IOT)*, which can be attached only to an SQL view that is deemed to be read-only. When the IOT is activated, **INSERT**, **UPDATE**, or **DELETE** statements no longer fail when referencing the view. The IOT intercepts the insert, update, or delete event and redirects the action to the appropriate base table.

For more information, see *Stored Procedures, Triggers, and User-Defined Functions on DB2 Universal Database for iSeries*, SG24-6503.

8.4.3 User-defined functions

DB2 for i provides a rich set of database functions that you can use. For example, there are functions to get the month, day, or year from a time stamped column. However, there are occasions when a custom function is needed to do some specific task. That is where user-defined functions become valuable. User-defined functions (UDF) give you the ability to extend DB2 with your own custom built SQL functions. UDFs are part of the DBMS and give you the ability to use them inside SQL statements. There are two types of UDFs:

- ▶ SQL UDFs
- ▶ External UDFs

Each UDF type can return one of two different pieces of data.

- ▶ Scalar UDF: Returns a single value.
- ▶ Table UDF: Returns a table of values.

For more information about user-defined functions, see *Stored Procedures, Triggers, and User-Defined Functions on DB2 Universal Database for iSeries*, SG24-6503.

8.5 Modern reporting and analysis

So far, this chapter has described techniques for performing modern database access. This section looks at modern tools for your database queries, reporting, and analysis. Here are some reasons that this topic is important:

- ▶ The queries with Query/400 do not use the SQE optimizer.
- ▶ Query/400 is only green screen 5250 based.
- ▶ Query/400 reports are simple and you cannot include graphics or other dynamic elements.
- ▶ Many users need simple graphic reports that can be easily manipulated. This requirement is especially true when dealing with statistical data.
- ▶ Query/400 does not use UDF.

Business Intelligence (BI) is a broad term that relates to applications that are designed to analyze data for purposes of understanding and acting on the key metrics that drive profitability in an enterprise. Key to analyzing that data is providing fast, easy access to it while delivering it in formats or tools that best fit the needs of the user.

At the core of any BI solution are user query and reporting tools that provide intuitive access to data that supports a spectrum of users from executives to power users, from spreadsheet aficionados to the external Internet consumer.

IBM DB2 Web Query for i offers a set of modernized tools for a more robust, extensible, and productive reporting solution than the Query/400 product. DB2 Web Query for i preserves investments in the reports that are developed with Query/400 by offering a choice of importing definitions into the new technology or continuing to run existing Query/400 reports as is. But it also offers significant productivity and performance enhancements by using the latest in DB2 for i query optimization technology.

The DB2 Web Query for i product is a web-based query and report writing product that offers enhanced capabilities over the IBM Query/400 product. IBM DB2 Web Query for i includes technology to assist customers in their transition to DB2 Web Query. It offers a more modernized, Java based solution for a more robust, extensible, and productive reporting solution.

DB2 Web Query can query or build reports against data that is stored in DB2 for i (or Microsoft SQL Server) databases through browser-based user interface technologies.

You can build new reports with ease through the web-based, ribbon-like InfoAssist tool that uses a common look and feel that can extend the number of personnel that can generate their own reports.

You can simplify the management of reports by reducing the number of report definitions that are required by using parameter-driven reports. You can deliver data to users in many different formats, including directly into spreadsheets, or in boardroom-quality PDF format, or viewed from the browser in HTML. Use advanced reporting functions such as matrix reporting, ranking, color coding, drill-down, and font customization to enhance the visualization of DB2 data.

DB2 Web Query offers features to import Query/400 definitions and enhance their look and functions. It enables you to add OLAP-like slicing and dicing to the reports or to view reports in disconnected mode for users on the go.

In addition to creating and viewing modern graphical reports, this can now also be done from your mobile device.

Some examples of some of the advantages of DB2 Web Query are shown in Figure 8-23.



Figure 8-23 DB2 Web Query advantages

For more information, see *IBM DB2 Web Query for i Version 2.1 Implementation Guide*, SG24-8063.

8.5.1 Removing OPNQRYF and RUNQRY

Before SQL was available on the IBM i, using the IBM OS/400® operating-system Open Query File (**OPNQRYF**) command (now an IBM i operating system command) was a popular method of accessing information that was stored in a database table.

As SQL does on IBM i today, **OPNQRYF** provided midrange application developers with a great deal of flexibility and ease of use when specifying selection, joining, ordering, and grouping. Instead of hardcoding various access methods within the tiers of multiple-case or if-then-else statements, programmers simply constructed character strings dynamically with these criteria and passed those strings as **OPNQRYF** parameters. The database engine processed that request accordingly to return a result set. The dynamic nature of this command eliminated many lines of code and made applications easier to understand and maintain.

OPNQRYF is still widely used on the IBM i platform today. Thousands of applications take advantage of its ability to dynamically access information that is stored in an IBM DB2 for i database. There are many ways that you can use **OPNQRYF** in application programs. Some of the most popular **OPNQRYF** uses are listed here:

- ▶ Dynamic record selection
- ▶ Dynamic ordering without using data description specification (DDS)
- ▶ Grouping (summarizing data)
- ▶ Specifying key fields from different files
- ▶ Dynamic joining without using DDS
- ▶ Unique-key processing
- ▶ Defining fields that are derived from existing field definitions
- ▶ Final total-only processing
- ▶ Random access of a result set
- ▶ Mapping virtual fields

Although **OPNQRYF** does an excellent job at performing these types of tasks and IBM still fully supports it, you can use a pure SQL-based implementation to receive additional benefits while maintaining the application flexibility that is provided by **OPNQRYF**. In fact, there are many reasons to consider converting your applications to use pure SQL access. The following sections examine some of these reasons in detail, including some approaches to performing this data-access conversion. There are many examples and other considerations to help you understand and carry out this conversion process successfully.

8.5.2 Reasons to consider conversion

Although **OPNQRYF** is adept at handling the tasks that are listed in 8.5.1, “Removing **OPNQRYF** and **RUNQRY**” on page 286, there are many compelling reasons to consider converting your programs to a pure SQL-statement model, including the following reasons:

- ▶ SQL is the industry standard.
- ▶ SQL is the strategic interface for DB2 for i.
- ▶ The SQL query engine (SQE) provides superior performance.
- ▶ The SQL interfaces offer advanced functions.
- ▶ DB2 performance tools are tailored for SQL interfaces.
- ▶ SQL offers simpler application development and maintenance.
- ▶ Using SQL reduces the number of programs and lines of code.
- ▶ Using SQL allows you to monitor your system activity in a more accurate manner.

8.5.3 SQL is the industry standard

Several years after **OPNQRYF** was made available on IBM System/38 servers, IBM shipped SQL as an application development tool for creating and accessing database objects on AS/400. Since that time, SQL has become the widely adopted industry standard for all relational database platforms. Although functional and powerful, **OPNQRYF** is a proprietary, non-SQL interface and is supported only on IBM midrange systems. In this evolving world of system openness, this is important: the days of relying on your green-screen application to provide the sole access to your data are nearing the end. Almost every non-5250 client application that accesses information from a database on the IBM i platform uses an SQL interface.

8.5.4 SQE features

There are multiple advantages when the SQE processes a query. Some of the major features that are only available with SQE are as follows:

- ▶ SQE plan cache

This internal, matrix-like repository holds access plans for queries that the SQE optimizes and allows plans for identical and similar statements to be shared across jobs. Reusing existing access plans means that the optimizer does not have to create plans, saving time and resources.

- ▶ Automated collection of column statistics

These statistics provide a valuable source of information to the optimizer when evaluating the cost of each available access plan. A better understanding of the data and more accurate estimates of the number of rows that are to be processed results in the selection of a better access plan and a more efficient query.

- ▶ Autonomic indexing

This is the ability of the optimizer to create temporarily maintained indexes that both the current query and future queries can use, system-wide.

8.5.5 Superior performance of SQE

SQE also introduced new data-access primitives to process the data. The improved results are most evident for complex queries that require grouping, ordering, and joining. As a result, SQL statements that use SQE generally perform better than similar requests that CQE processes. Because CQE processes all **OPNQRYF** requests, most SQL-access requests to the database should outperform an equivalent **OPNQRYF** request.

Quantifying how much better an application performs after making modifications is a tricky venture. Many factors influence performance, so it is better to conduct your own benchmarks, perform the conversion on a sample application, and record the resulting performance metrics.

To learn more about SQE, see Redbooks publication *Preparing for and Tuning the SQL Query Engine on DB2 for i5/OS*, SG24-6598.

8.5.6 Advanced functions of the SQL interface

SQL on DB2 for i is rich with features that are simply not available with **OPNQRYF**. The following list highlights some of the more prominent features that can be used only with pure SQL data-access methods. Here are some of the more prominent features to consider:

- ▶ UDFs

OPNQRYF provides a base set of built-in functions to use in query selection and grouping and for defining a derived field. However, **OPNQRYF** does not always have the required functions for you to implement complex business calculations or processes, such as computing shipping costs or a customer's credit risk. If you need to extend **OPNQRYF** by writing custom functions, there is no way to do so. With SQL, it is easy to create UDFs and to use them just as you use the SQL built-in functions.

- ▶ User-defined table functions (UDTFs)

Another valuable SQL feature is the ability to search for and retrieve data in non-relational system objects (such as data areas and data queues, and even information that is stored in the integrated file system (IFS)), and then return that information to any SQL interface. You can do this by creating UDTFs.

- ▶ Materialized query tables (MQTs)

This is a summary table that contains the results of a previously run query, along with the query's definition. It provides a way to improve the response time of complex SQL queries. What sets an MQT apart from a regular temporary summary table is that the SQE optimizer is aware of its relationship to the query and base tables that are used to create and populate it. This means that the optimizer considers using the MQT in the access plan of subsequent similar queries. Because the MQT is already created and populated, this can improve performance for complex queries.

For more information about MQTs, see *Creating and using materialized query tables (MQT) in IBM DB2 for i*, found at:

<http://www.ibm.com/partnerworld/wps/servlet/ContentHandler/SR0Y-6UZ5E6>

- ▶ SQL views

An SQL view is a virtual table. It provides another way to view data in one or more tables and is created based on an SQL **SELECT** statement. With views, you can transform the data and move business logic from the application layer down to the database. You can define selection, grouping, and complex logic through **CASE** statements. This means that, in an SQL view, you can define the selection, joining, and ordering specifications for an **OPNQRYF** command. Because those specifications exist in the view definition, you do not need to specify them again in SQL. Instead, the SQL refers to the view. In fact, a technique that many System i programmers employ is to specify an SQL view in the **OPNQRYF FILE** parameter.

- ▶ Subqueries

By definition, a subquery is a query that is embedded within the **WHERE** or **HAVING** clause of another SQL (parent) statement. The parent statement then uses the rows that are returned to further restrict the retrieved rows. A subquery can include selection criteria of its own, and these criteria can also include other subqueries. This means that an SQL statement can contain a hierarchy of subqueries. This is a powerful SQL searching mechanism that cannot be accomplished with **OPNQRYF**.

- ▶ Common table expressions (CTEs) and recursive SQL

CTEs can be thought of as temporary views that exist only when running an SQL statement. After the CTE is defined, you can reference it multiple times in the same query. You can use this to reduce the complexity of the query, making it easier to comprehend and maintain.

Among the IBM i enhancements for DB2 for i is the ability for a CTE to reference itself. This feature provides the mechanism for recursive SQL, which is especially useful when querying data that is hierarchical in nature (such as a bill of materials, organizational charts, and airline flight schedules). For more information about recursive CTEs, see *IBM i V5R4 SQL Packs a Punch*, found at:

http://www.ibm.com/systems/resources/systems_i_software_db2_pdf_rcte_olap.pdf

- ▶ Complex joining

To help solve your complex business requirements, SQL provides various ways of joining data in tables together by using **INNER**, **OUTER**, and **EXCEPTION** joins.

- ▶ Fullselect

An SQL fullselect is the term for generating an SQL result set by combining multiple **SELECT** statements that use the **UNION**, **INTERSECT**, and **EXCEPT** operators. This is another feature that helps you solve more complex business requirements and is beyond the capabilities of **OPNQRYF**.

- ▶ Encryption

The importance of data security in today's business environment cannot be overstated. Hackers, phishers, and others with malicious intentions constantly and incessantly attempt to access data to which they have no rights. If you store sensitive information in your database, it is your obligation (and often a lawful requirement) to protect that information from these threats. Object security, firewalls, and other security measures are all valuable tools to thwart unauthorized access and to secure the data. Data encryption provides another line of defense. A hacker who can penetrate your security implementations finds only encrypted information.

If you need more detail about this concept, review the following documents:

- ▶ *Moving from OPNQRYP to SQL*, found at:
<http://public.dhe.ibm.com/partnerworld/pub/whitepaper/13d32.pdf>
- ▶ *Introduction to DB2 for i query optimization*, found at:
<http://www.ibm.com/partnerworld/wps/servlet/ContentHandler/servers/enable/site/education/ibp/dada/>
- ▶ *Database performance and query optimization*, found at:
<http://pic.dhe.ibm.com/infocenter/series/v7r1m0/topic/rzajq/rzajq.pdf>

8.6 Database tools

When you are working with your database, there are many tools that are available that can improve your ability to manage and modernize your database. This section describes some of these tools:

- ▶ IBM Data Studio
- ▶ IBM InfoSphere Data Architect
- ▶ IBM i Client Navigator

Table 8-1 shows a comparison of these tools.

Table 8-1 Comparison of database tools

Features	System i Navigator 7.1	IBM Data Studio 4.1	InfoSphere Data Architect 9.1
Compatibility with IBM i 7.1	Yes	Yes	Yes
SQL Routine and PL Development	Yes	Yes	Yes
SQL Routine Graphical Debugger	Yes	Yes ^a	Yes ^a
Administration	Yes	Yes	Yes
Visual Explain	Yes	No ^b	No ^b
Reverse Engineering	Yes	Yes	Yes
Physical Data Modeling	No ^c	Yes	Yes
Logical Data Modeling	No	No	Yes
Eclipse based	No	Yes	Yes
Integrates with Rational	No ^d	Yes	Yes
Supports Multiple DBMS	No	Yes	Yes
Source Change Management	No	Yes ^e	Yes ^e

a. Requires DB2 Connect

b. Not supported for IBM i

c. Has database mapping capability

d. Can execute SQL scripts that are created in IBM Data Studio or IBM InfoSphere Data Architect

e. Integrates with Rational tools or any Eclipse-based open source product

8.6.1 Managing DB2 for i with IBM Data Studio

In this section, you learn how to manage DB2 for i with IBM Data Studio. This is a powerful tool for accessing your IBM i database. You can use Data Studio to create and manage database tables. You can also create graphical views to easily see the format and layout of your database.

Start IBM Data Studio through one of the following methods:

- ▶ Windows: Click **Start** → **All Programs** → **IBM Data Studio** → **IBM Data Studio 4.1.0.0 client**.
- ▶ Linux: Select **Applications** → **IBM Data Studio** → **Data Studio 4.1.0.0 full client**.

When you open the product, you can specify what workspace you want to use. Specify the workspace name, as shown in Figure 8-24.

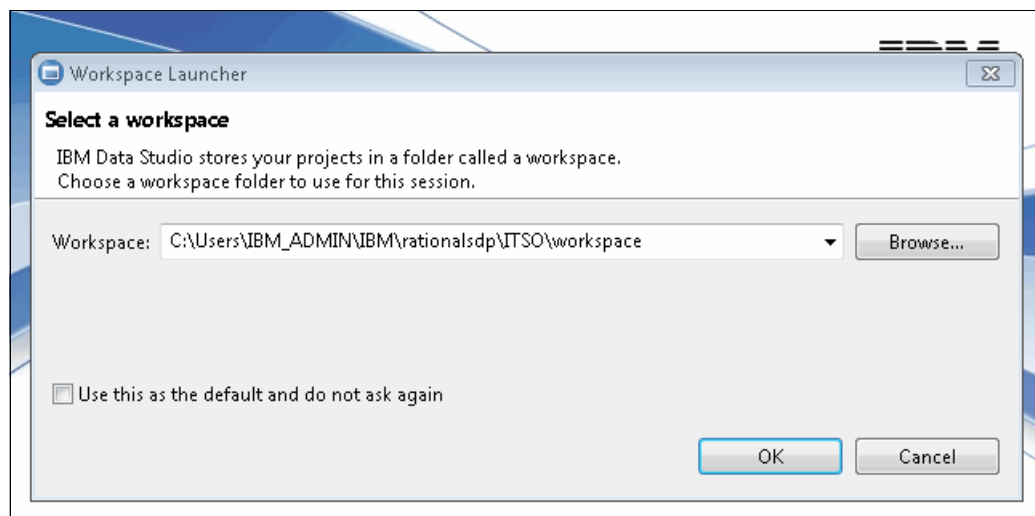


Figure 8-24 Specifying the workspace name

Note: A workspace is a location to save your work, customizations, and preferences. Your work and other changes in one workspace are not visible if you open a different workspace.

After you start the product, you see a window that is similar to the one that is shown in Figure 8-25. This is the default perspective for IBM Data Studio.

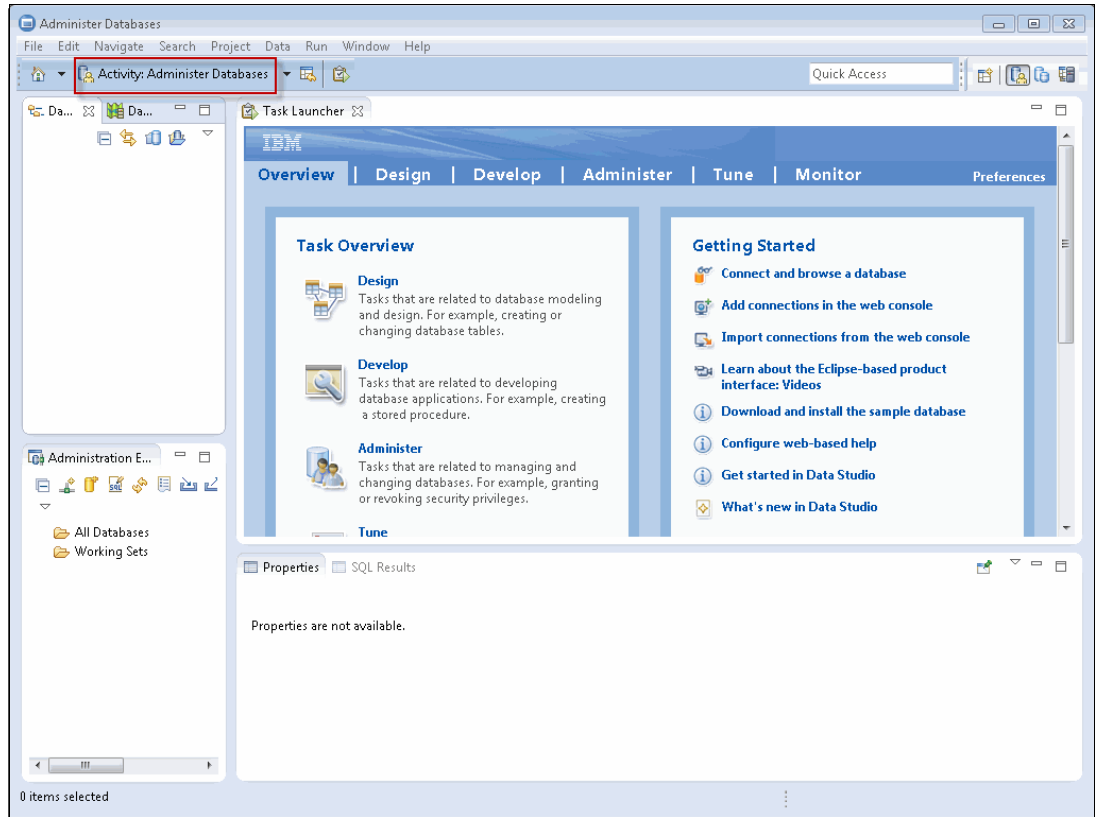


Figure 8-25 Data Studio initial window

Connecting to the database and filter

Because this is a tool for dealing with databases, we first illustrate how to connect to database and to diagram an existing model.

Complete the following steps:

1. Create a project. In the Project Explorer view, right-click and select **New** → **Project**, as shown in Figure 8-26.

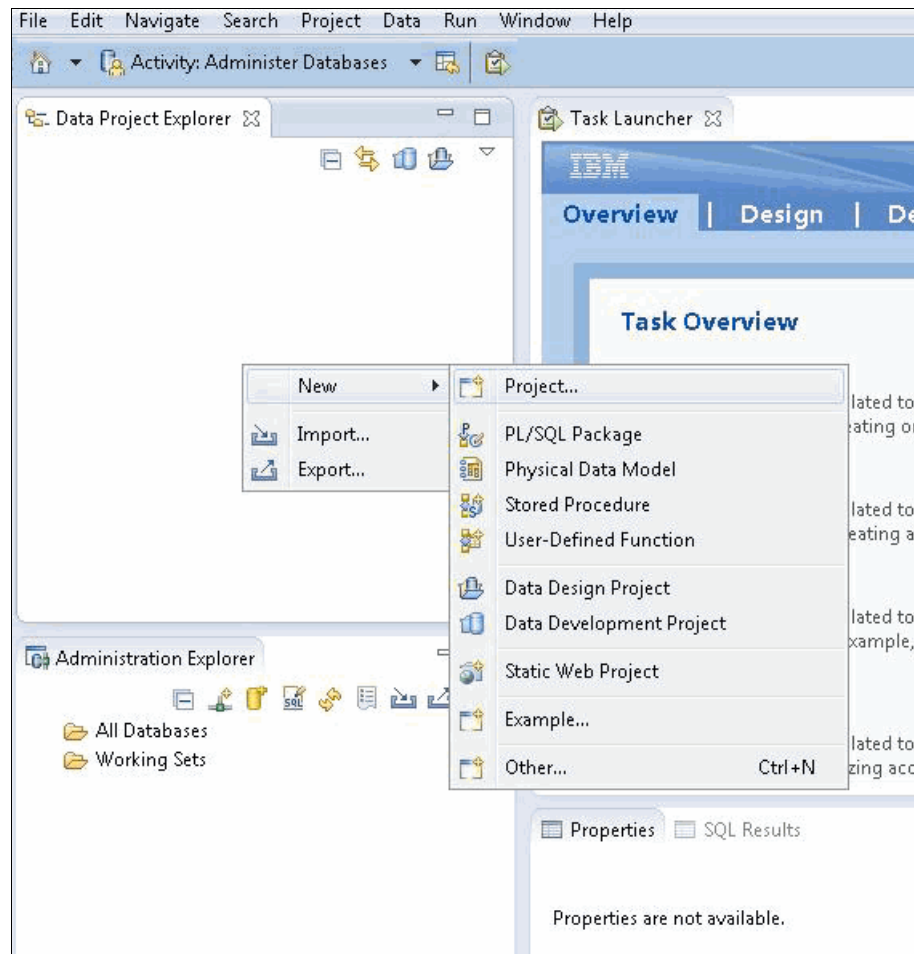


Figure 8-26 Creating a project

2. Indicate which type of project is being created. Select **Data Design Project**, as shown in Figure 8-27, and click **Next** to continue.

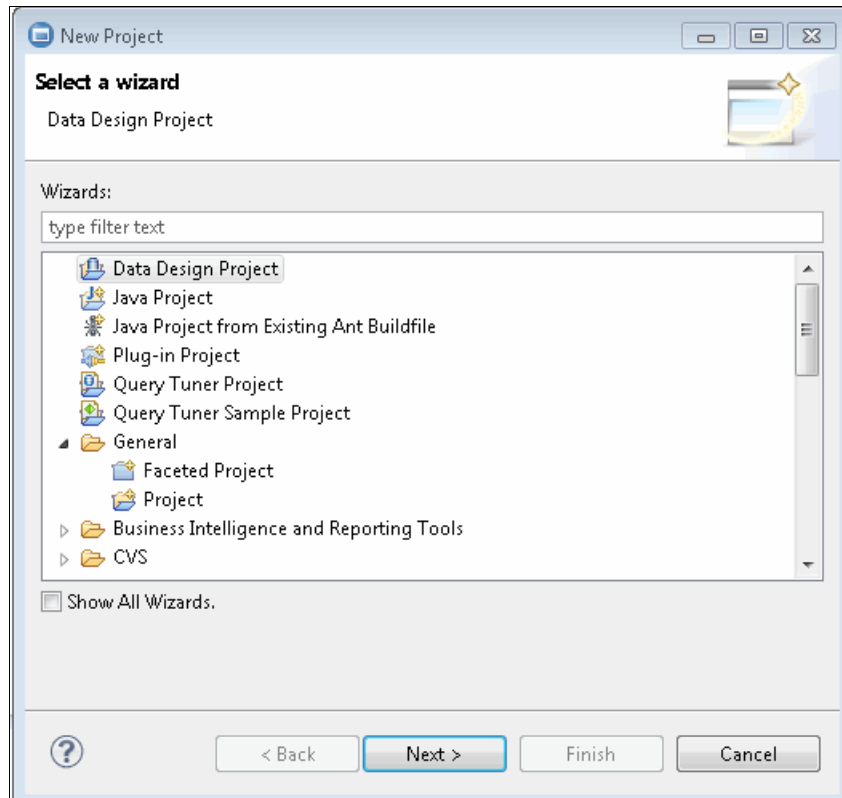


Figure 8-27 Specifying the project type

3. Enter the name of project, as shown in Figure 8-28 on page 295, and click **Finish** to continue.

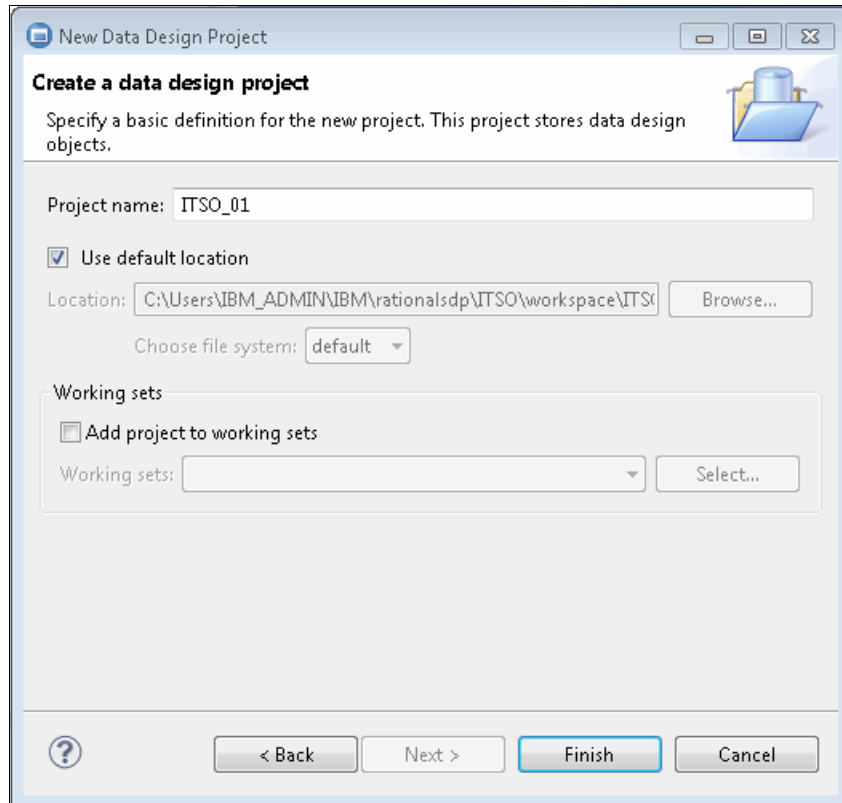


Figure 8-28 Specifying the project name

- Next, you can see the workspace window, as shown in Figure 8-29.

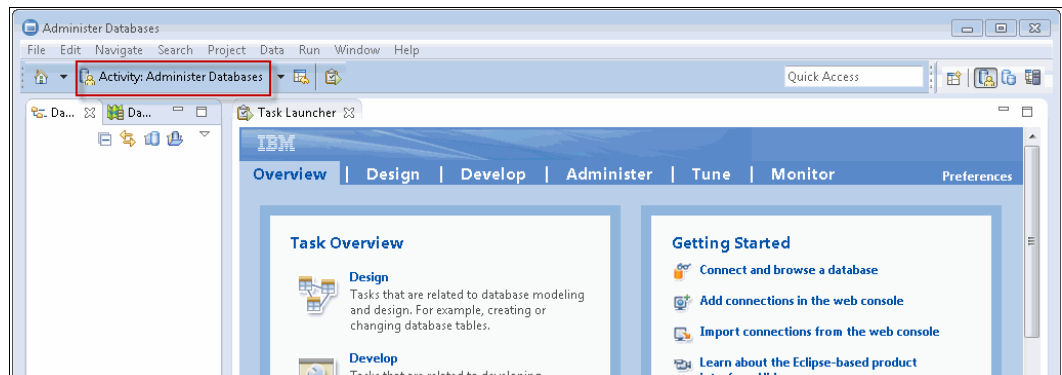


Figure 8-29 Main IBM Data Studio window with the created project

Now you have a workspace. Next, you must create a connection to the database.

5. Select the **Database Connections** option, right-click, and select **New**, as shown in Figure 8-30.

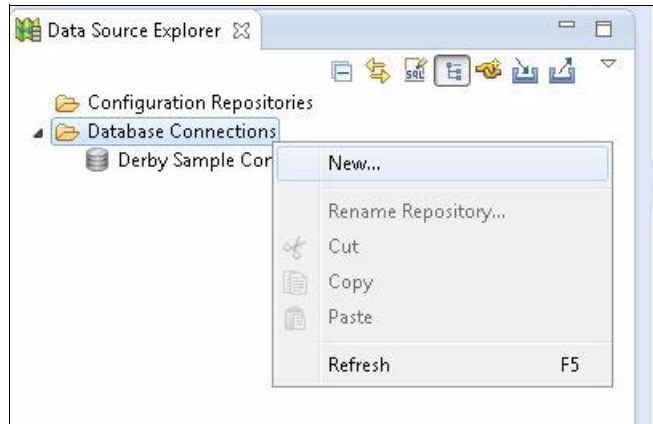


Figure 8-30 New database connection

The product asks you about the type of database. You must select **DB2 for i** and you should set the parameters for this connection as shown in Figure 8-31.

There are several mandatory fields:

- Host Name
- User Name
- Password

The Default schema field is optional.

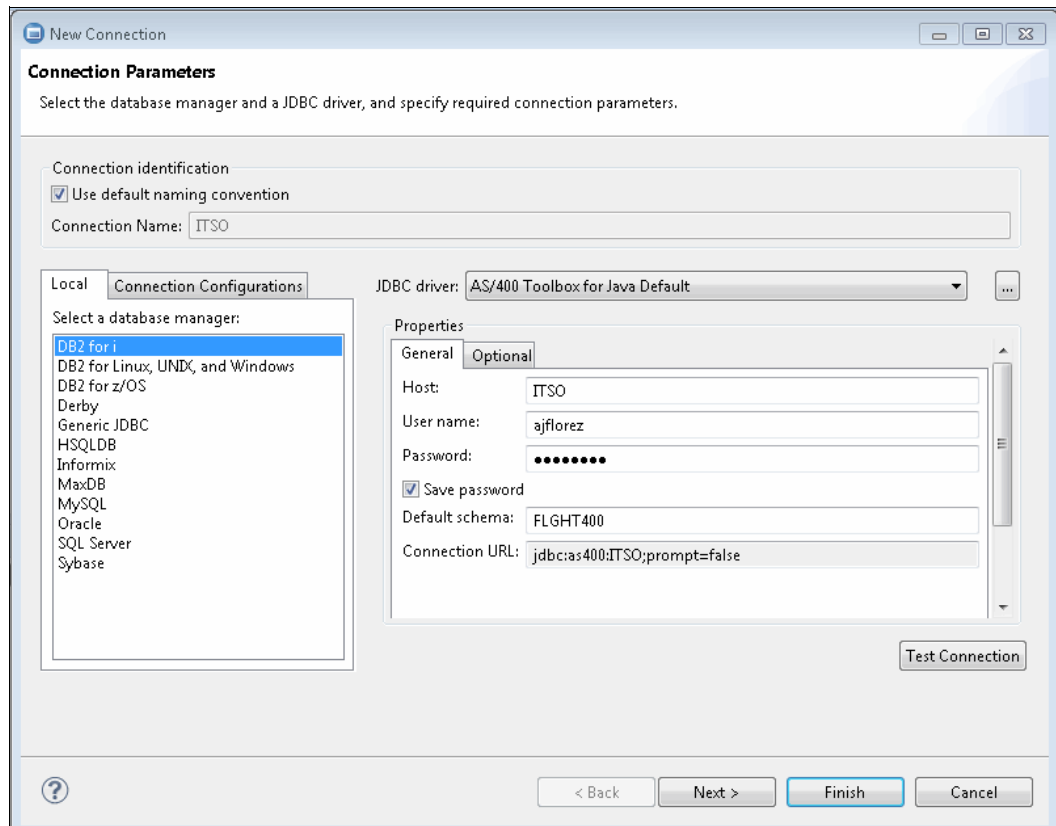


Figure 8-31 Setting the connections parameters

6. Press **Next** to continue. You see the New Connection window, as shown in Figure 8-32. In this window, you can link this connection to a physical model if you want. Otherwise, leave this space empty.

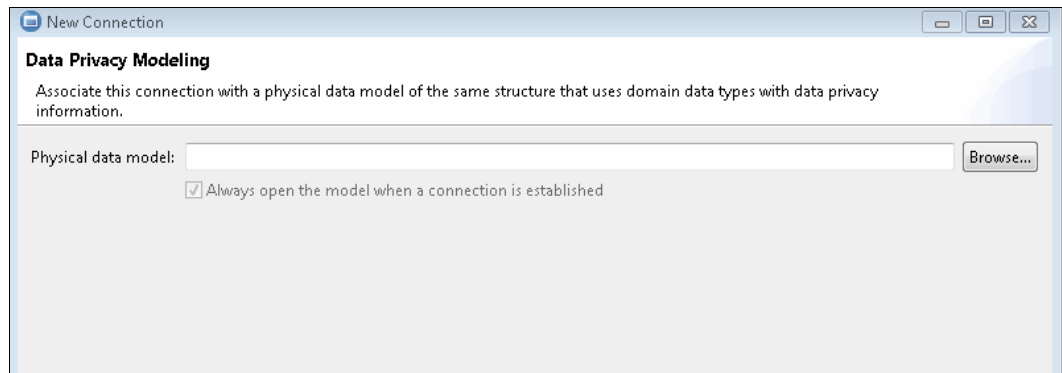


Figure 8-32 Data Privacy Modeling

7. Click **Next** to continue. Normally on the IBM i there are many schemas. You can filter to see only some schemas. In this window, you can filter the schemas by expression or by selection. It is important to select **Disable filter** to enable the option filters, as shown in Figure 8-33.

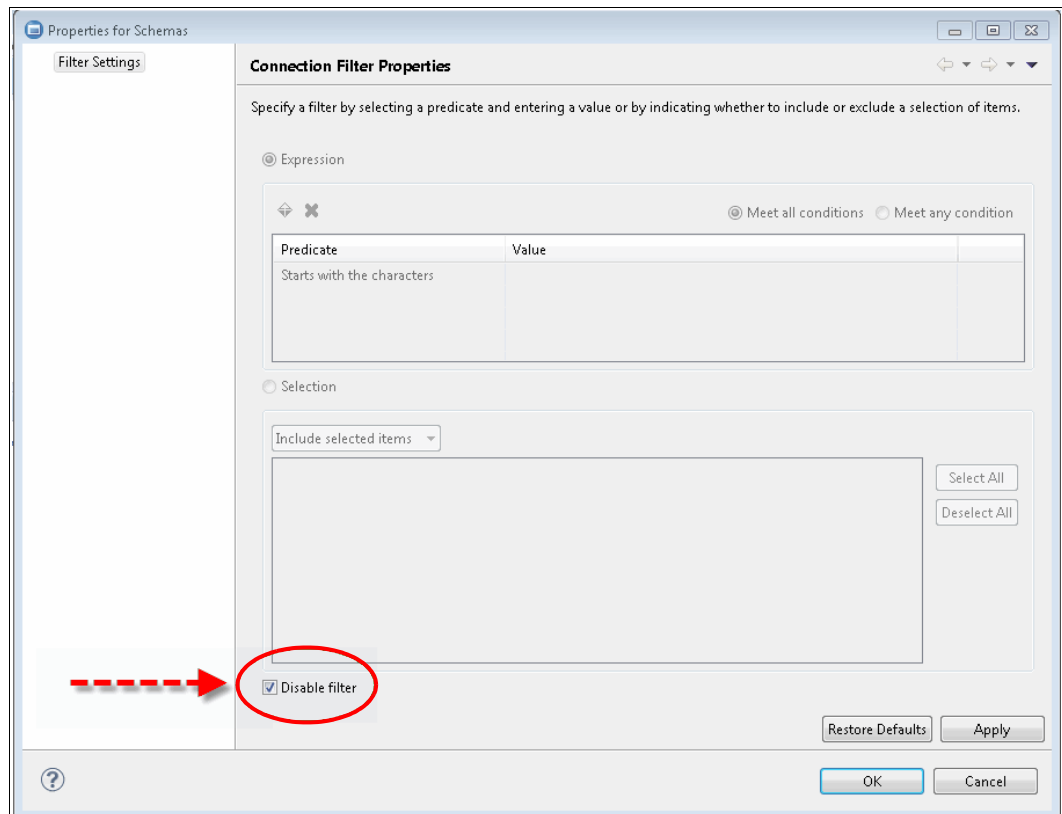


Figure 8-33 Filtering the list of schemas

You can filter by expression, as shown in Figure 8-34.

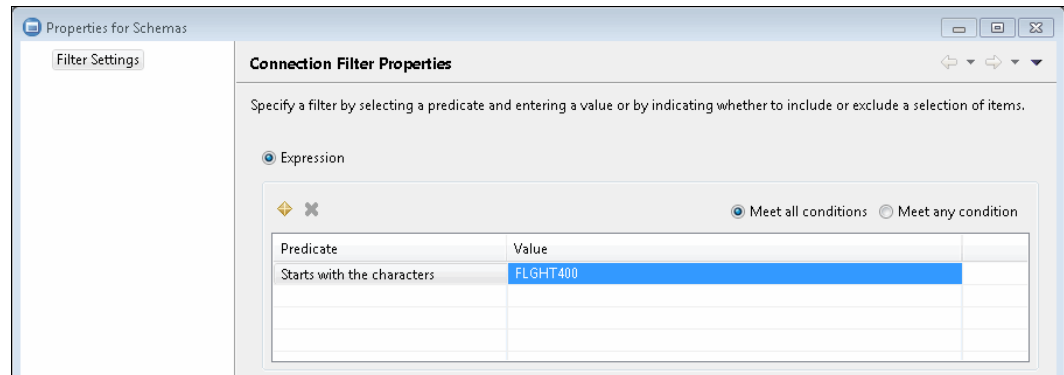


Figure 8-34 Filtering by expression

The other option is **Filter by selection**. You can check each schema and view it, as shown in Figure 8-35.

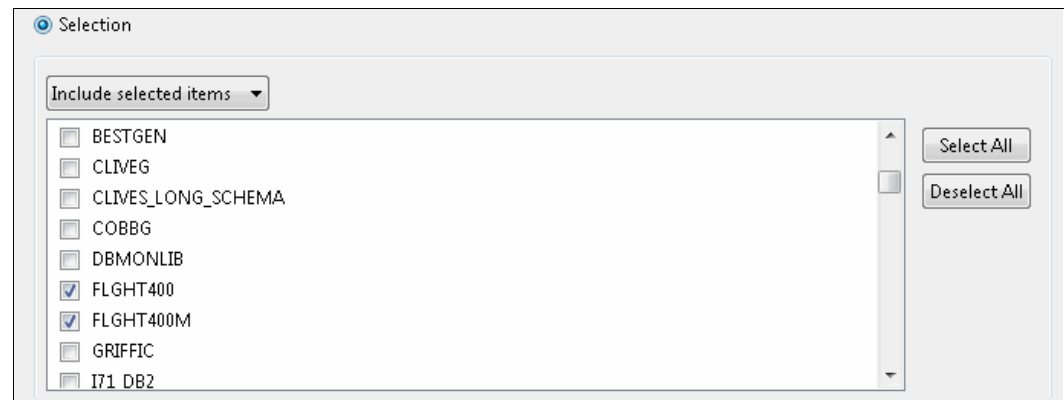


Figure 8-35 Filtering by selection

8. After you select the filters, click **Apply** and then **OK** to continue. You can see the connection that is defined in your workspace, as shown in Figure 8-36.

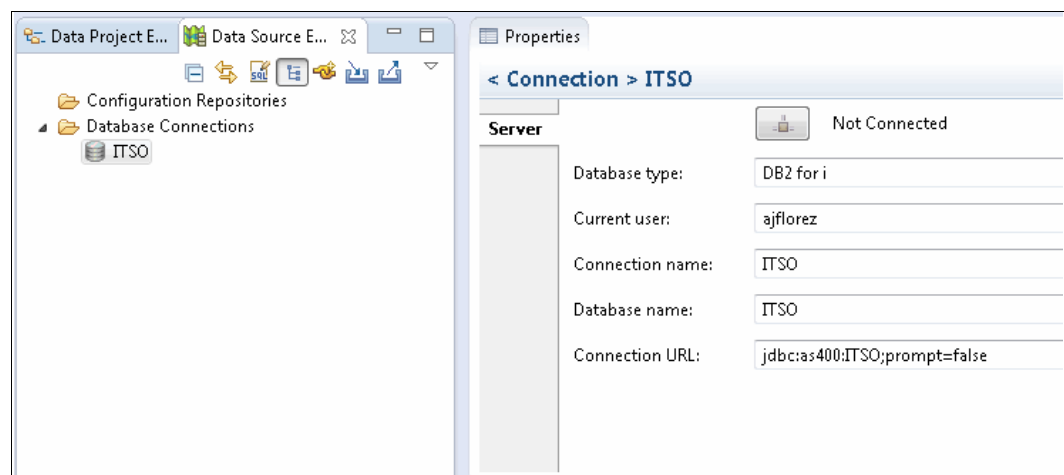


Figure 8-36 Connection defined

You are ready to work with DB2 for i from IBM Data Studio.

If you look in the Data Source Explorer view, you can see the connection tree. You can click to expand each item, as shown in Figure 8-37.

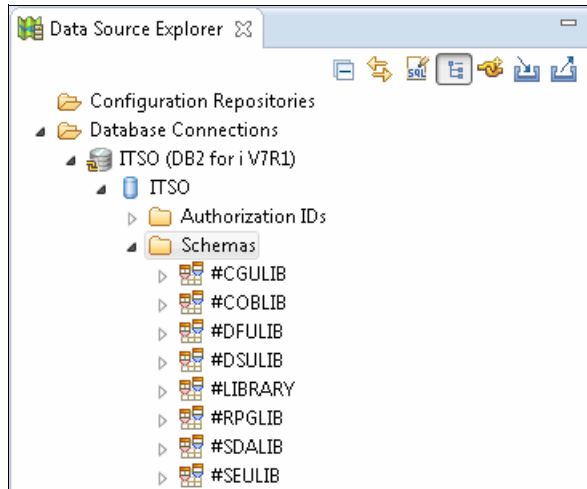


Figure 8-37 Connection tree

You can change the filter as needed. To change the filter, right-click the **Schemas** option, as shown in Figure 8-38.

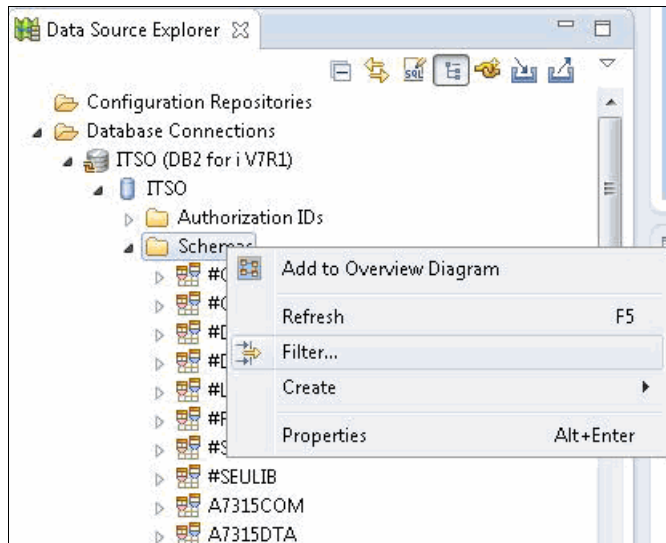


Figure 8-38 Right-clicking the Schemas option

9. Click **Apply** and **OK** to filter the schemas.

In this case, we select by expression. We want to view all schemas that start with the characters FLGHT400, as shown in Figure 8-39.

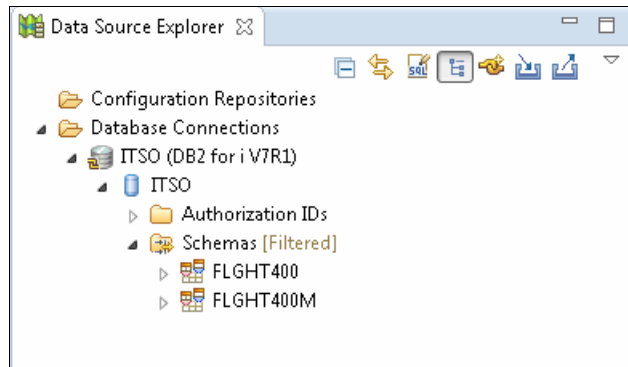


Figure 8-39 Results of the filter for FLGHT400

Adding an existing model to the diagram

IBM Data Studio enables you to diagram an existing model. To do so, complete the following steps:

1. Select the information to diagram. In this case, we select the tables to diagram. Right-click **Tables** and select the **Add to Overview Diagram** option, as shown in Figure 8-40.

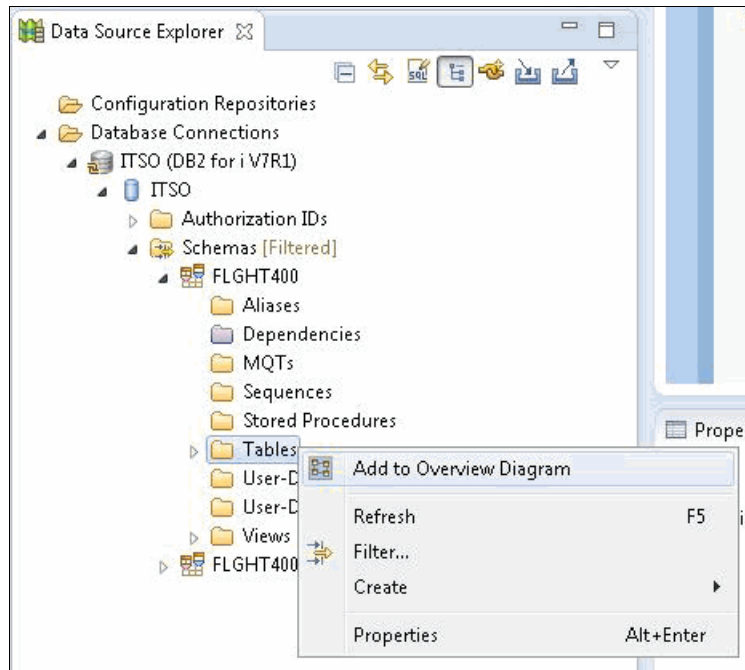


Figure 8-40 Selecting Add to Overview Diagram from the Tables folder

Now, you must select the tables to diagram. Select the check box next to each wanted table, as shown in Figure 8-41.

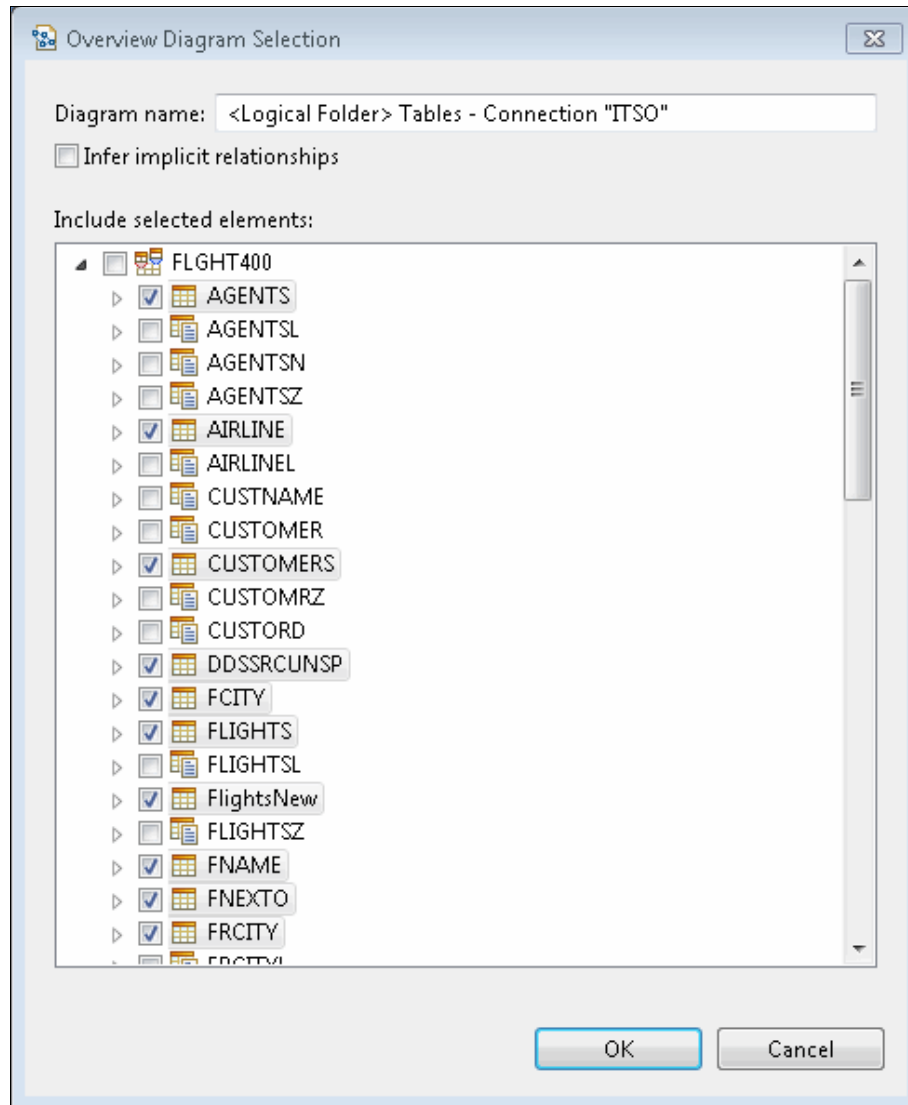


Figure 8-41 Selecting the tables to diagram

It is important to not select source physical files; they are not necessary, as shown in Figure 8-42.

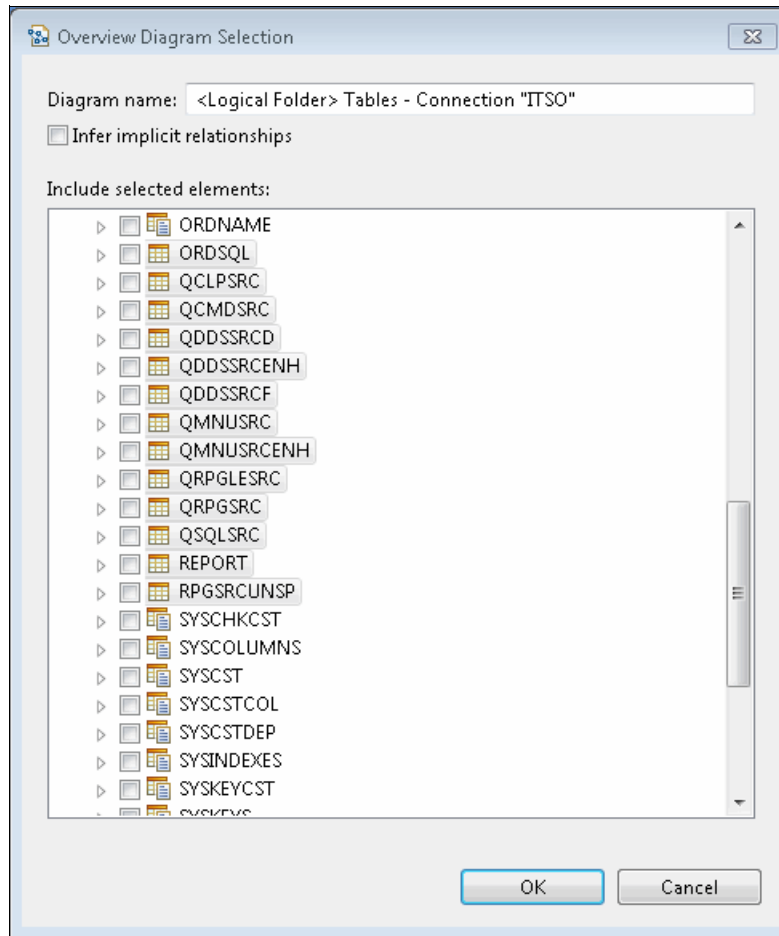


Figure 8-42 Source physical files are not selected

2. Click **OK** to continue. You can see a window, as shown in Figure 8-43. This is the diagram for the tables that are selected.

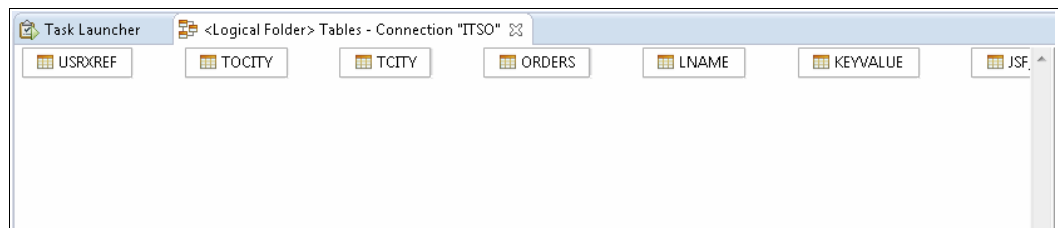


Figure 8-43 Logical model diagram

3. Now, you can personalize the view of the logical model. For example, you can show the relationships between the tables, as shown in Figure 8-44 on page 303.

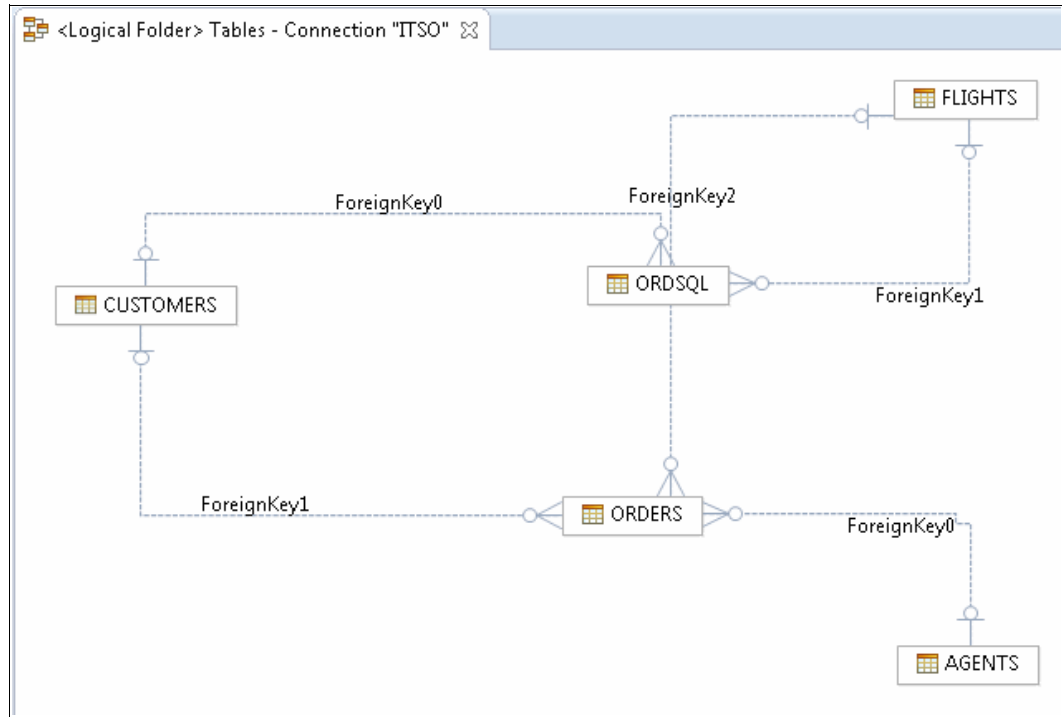


Figure 8-44 Relationships between tables

In Figure 8-45, you can see the outline view. You can navigate within the diagram for this view and you can locate part of the diagram.

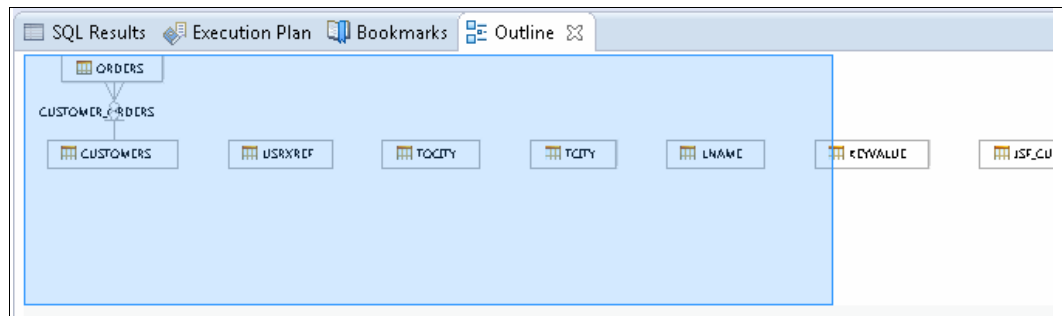


Figure 8-45 Outline view

When you are viewing the logical model, you can customize the view as needed, as shown in Figure 8-46.

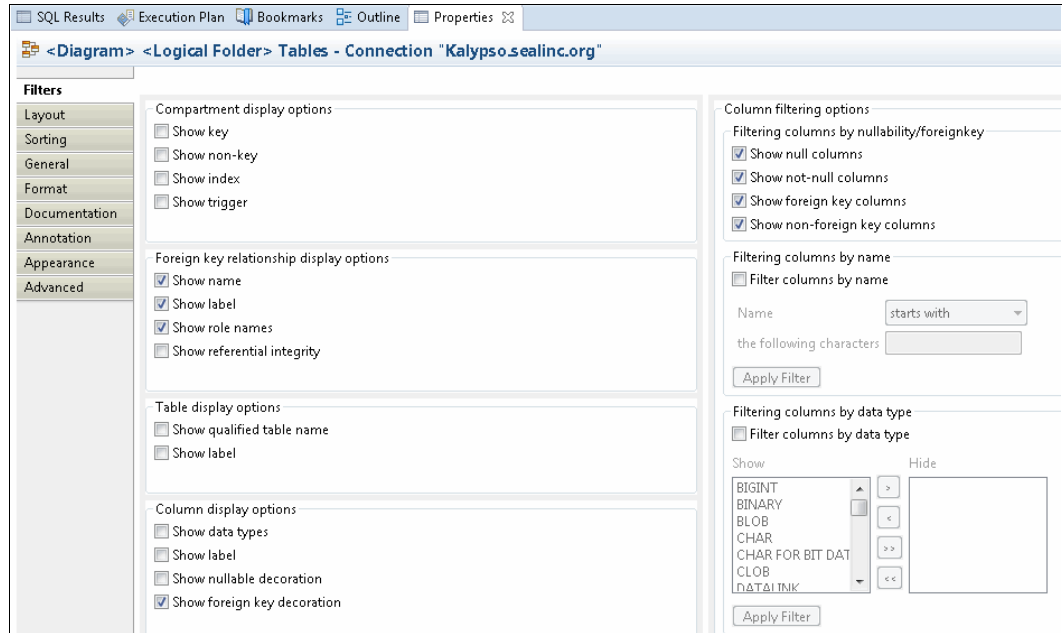


Figure 8-46 Properties of Logical Model View

Adding tables to the diagram

In this section, you add a table to the schema. It is assumed that you are connected to the database. Complete the following steps:

1. To add a table to diagram, you must expand the tree for the schema. When it is expanded, locate the table, right-click **Tables**, and select **Create** → **Table**, as shown in Figure 8-47.

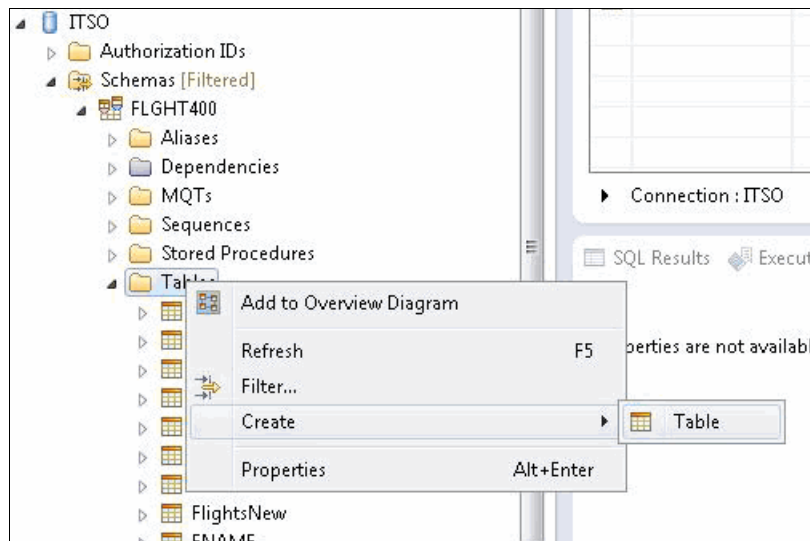


Figure 8-47 Creating a table

2. You see the Tables window, as shown in Figure 8-48 on page 305.

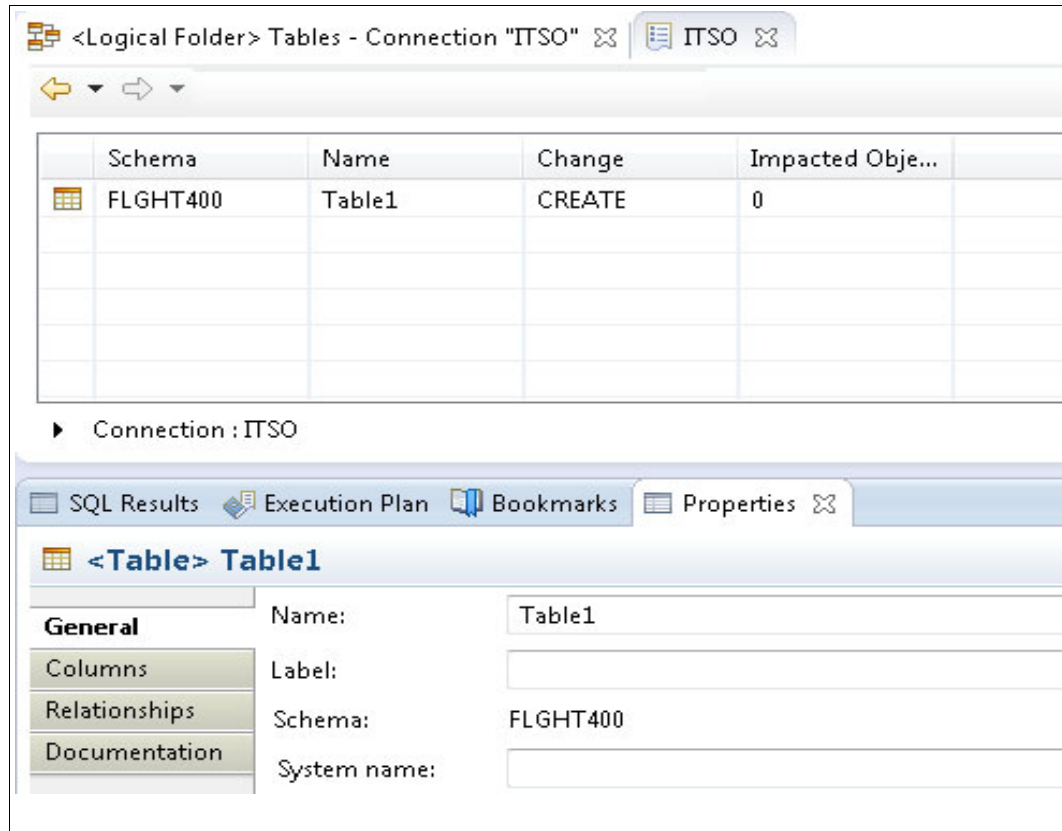


Figure 8-48 Table parameter view

3. Enter the table name and system name, as shown in Figure 8-49.

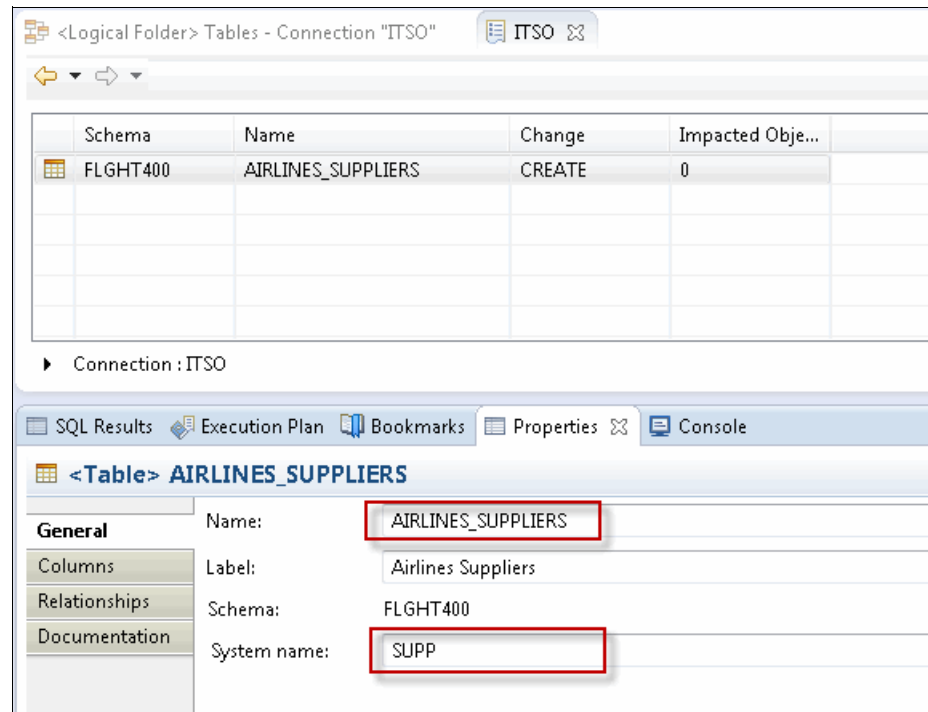


Figure 8-49 Setting the table parameters - table name and system name

- Now, you see the window that is shown in Figure 8-50. Here you can add attributes to the table. Click the **Column** tab and then click the **New** icon for each attribute that needs to be added.

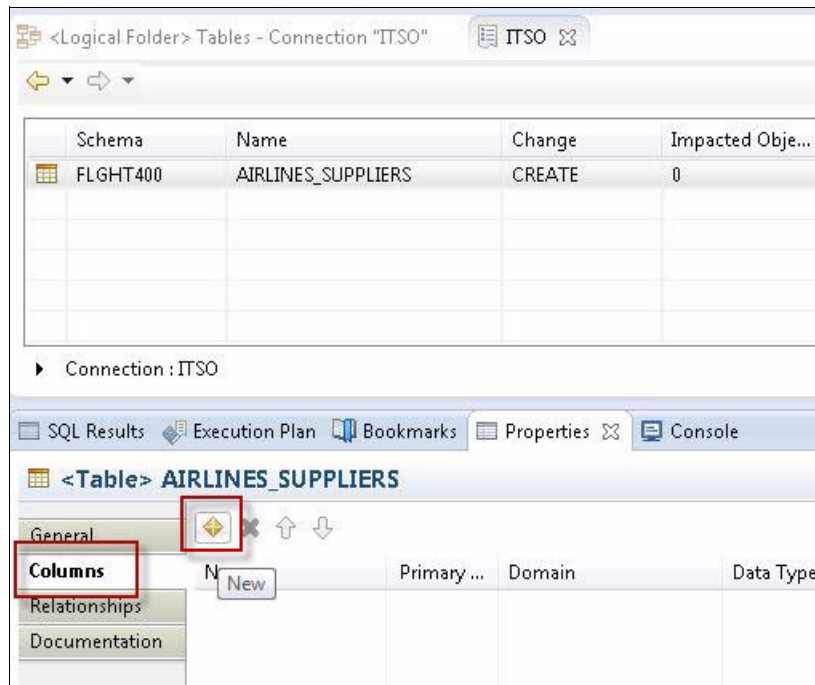


Figure 8-50 Adding attributes to a table

- The following attributes are added to the table. The circled icons in Figure 8-51 allow you to organize the table attributes. Here are the icons:
 - New field
 - Delete field
 - Up
 - Down

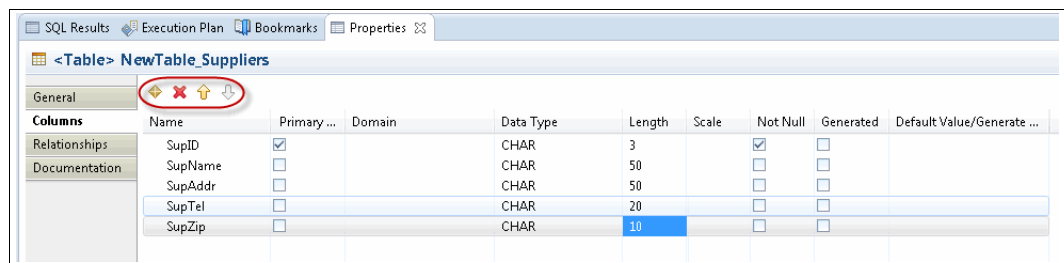


Figure 8-51 Table attributes and organization options

- When you have added all the fields, click the **Preview** icon. This icon is in the right corner of the window properties, as shown in Figure 8-52.

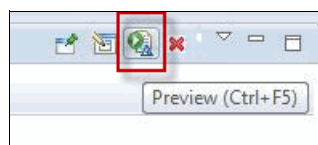


Figure 8-52 Preview icon

- The preview shows you the script for the table, as shown in Figure 8-53. Click **Run** to continue and create the table on the IBM i.

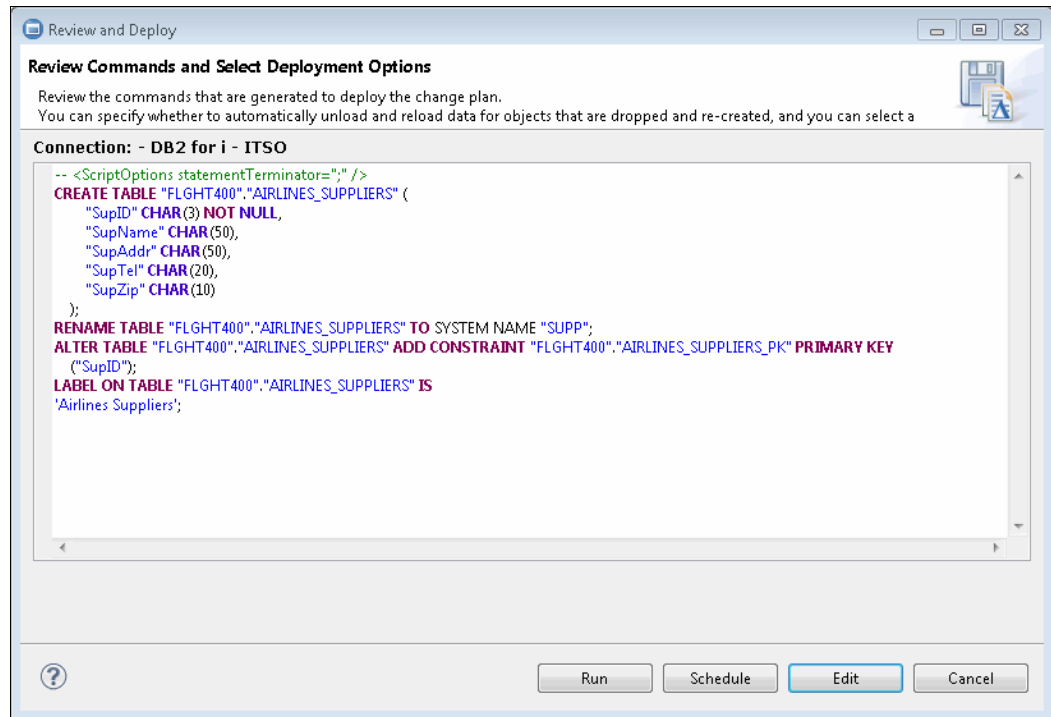


Figure 8-53 Script SQL for new table

Adding indexes to the diagram

This section walks you through the steps that are needed to add an index to the diagram. To accomplish this task, complete the following steps:

- To add an index to an existing model, you need to find the **Data Source Explorer** view in the **Tables** folder. In the **Tables** folder, find the table for the index. Right-click the **Indexes** folder and select **Create** → **Index** to continue, as shown in Figure 8-54.

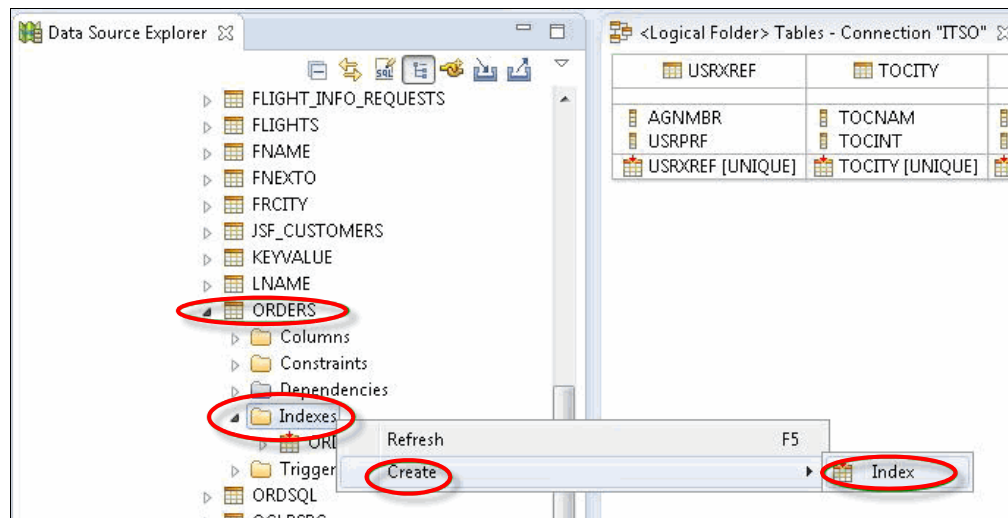


Figure 8-54 Adding the Index option to a diagram

2. Enter the name of the index in the Name field. In this case, we create ORDER_IDX1, as shown in Figure 8-55.

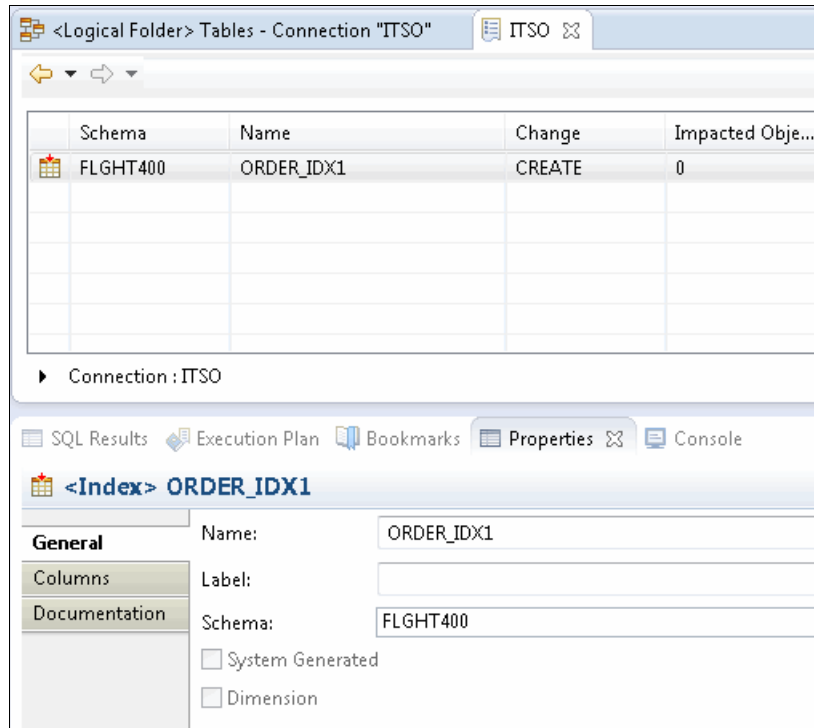


Figure 8-55 Index parameters

3. Next, click the **Columns** tab to add fields to this index. Click the **Sort Order** icon, as shown in Figure 8-56 on page 309. This allows you to specify the columns from this table that are added to this index. Select the wanted columns and click **OK** to continue.

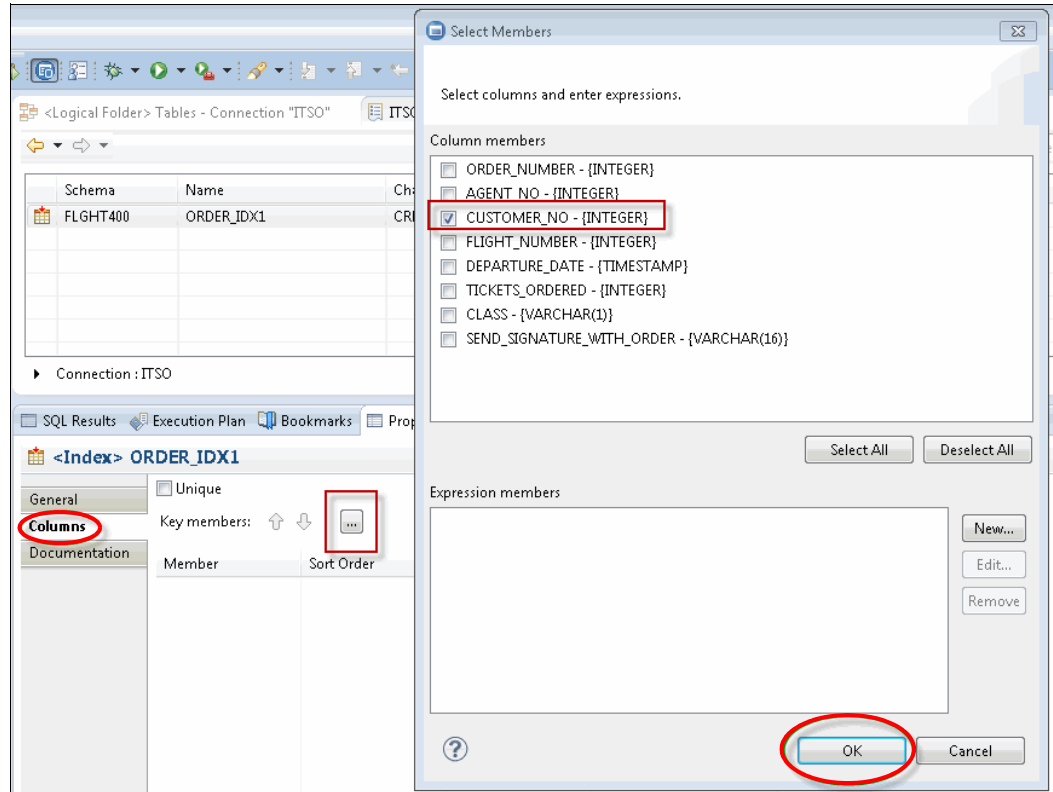


Figure 8-56 Selecting columns for adding an index

- Now, it is time to see what you have specified. In the upper right corner, click the **Review and Deploy** icon as shown in Figure 8-57.

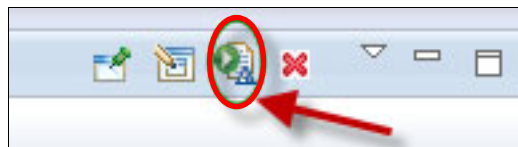


Figure 8-57 Clicking the Review and Deploy icon

5. Review the script for the new index and click **Run** to continue, as shown in Figure 8-58.

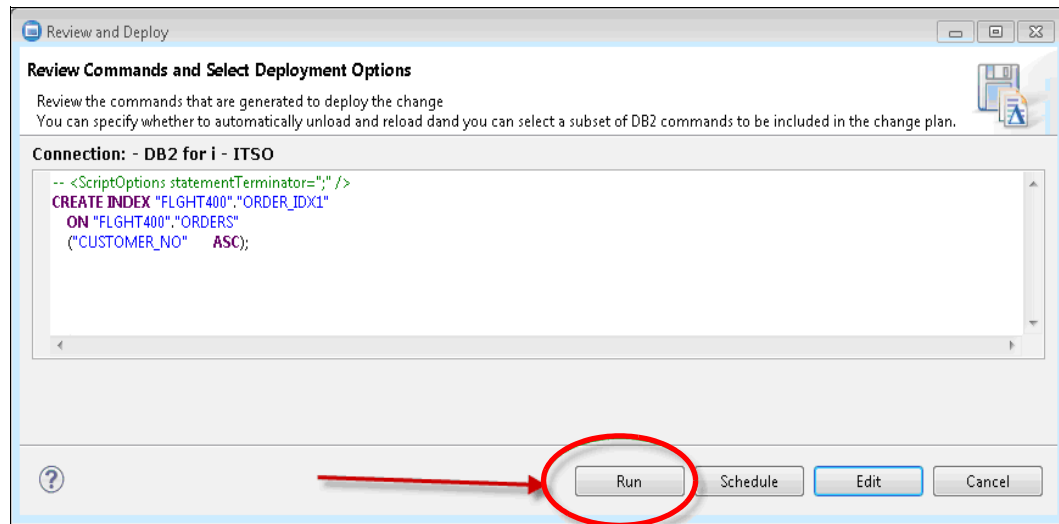


Figure 8-58 Review and Deploy script for new index

When the script is run, you see the new index in the Indexes folder, as shown in Figure 8-59.

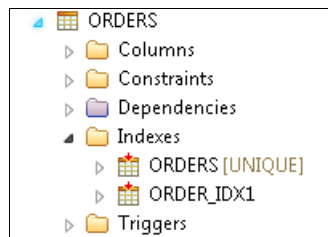


Figure 8-59 Index created

8.6.2 Managing DB2 for i with InfoSphere Data Architect

This section introduces the InfoSphere Data Architect tool. This is another powerful application for accessing your database.

Start IBM InfoSphere Data Architect by clicking **Start** → **All Programs** → **IBM InfoSphere** → **IBM InfoSphere Data Architect 9.1.0.0**, as shown in Figure 8-60.

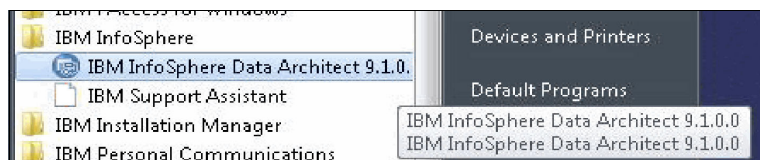


Figure 8-60 Starting InfoSphere Data Architect

When you open the product, you can specify what workspace you want to use. Specify the workspace name, as shown in Figure 8-61 on page 311.

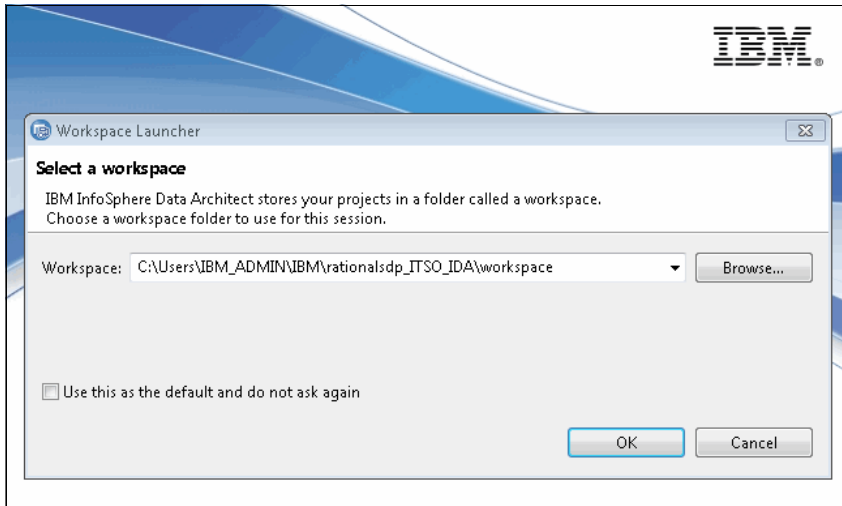


Figure 8-61 Specifying the workspace name for InfoSphere Data Architect

After you start the product, you see the main window that is shown in Figure 8-62. This is the default perspective for IBM InfoSphere Data Architect.

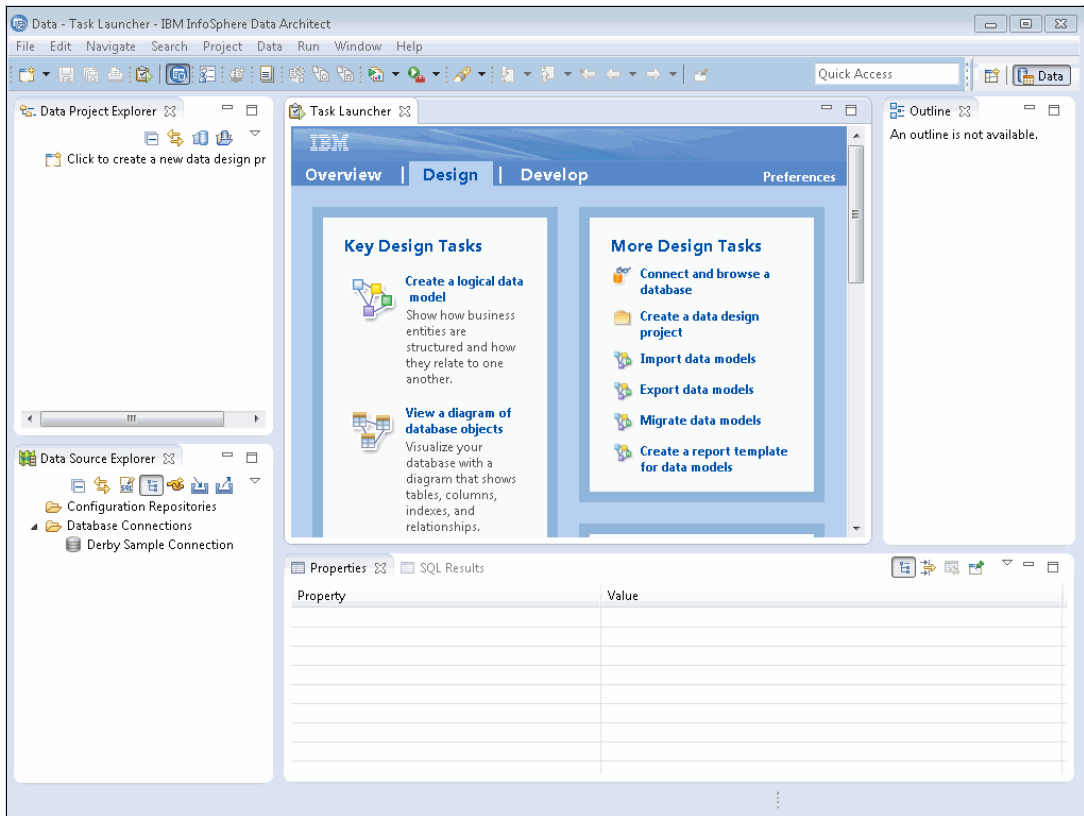


Figure 8-62 Default perspective for InfoSphere Data Architect

Connecting to the database and filtering

First, we show how to connect to a database and to diagram an existing model. Complete the following steps:

1. Create a project in the Project Explorer, view by right-clicking and clicking **New** → **Project**, as shown in Figure 8-63.

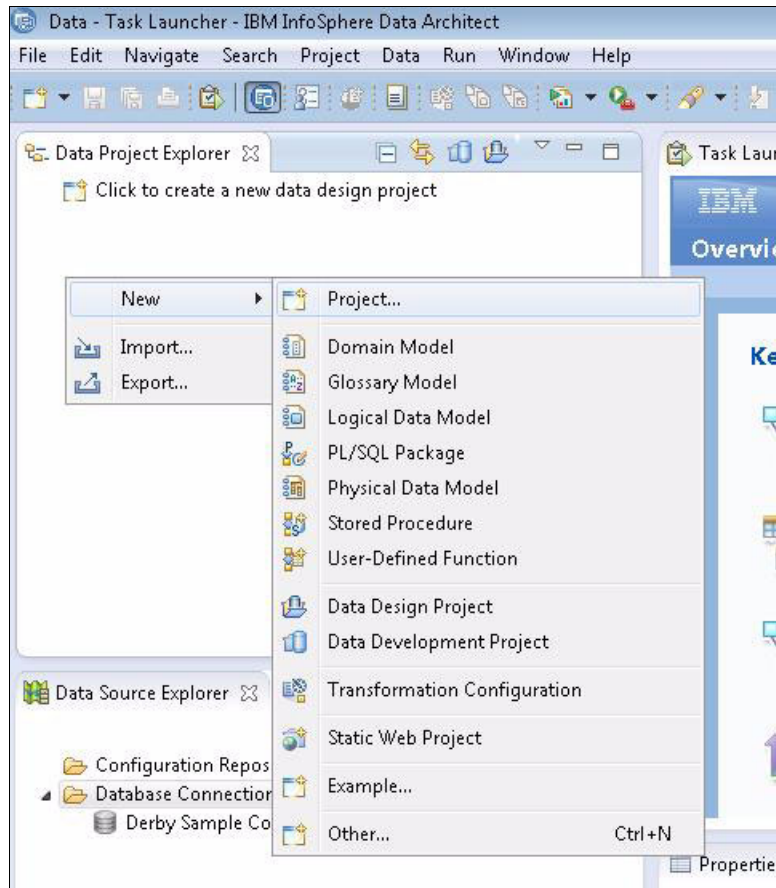


Figure 8-63 Selecting a new project in InfoSphere Data Architect

2. Indicate the type of project that is being created. For this example, we select the **Data Design Project**, as shown in Figure 8-64, and click **Next** to continue.

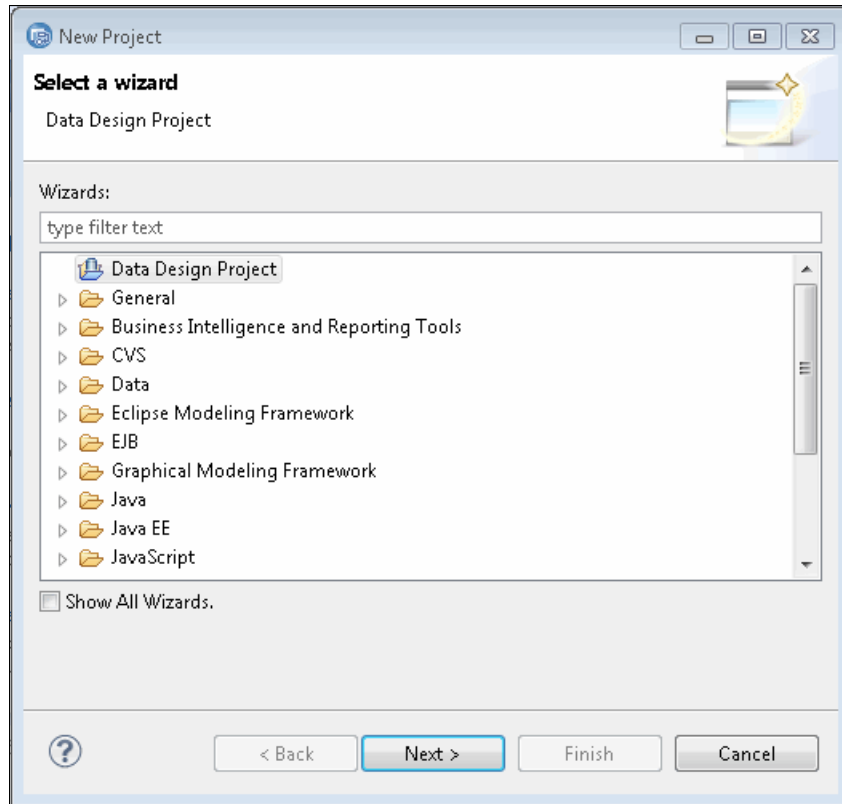


Figure 8-64 Selecting the type of project in InfoSphere Data Architect

3. Enter the project name, as shown in Figure 8-65, and click **Finish** to continue.

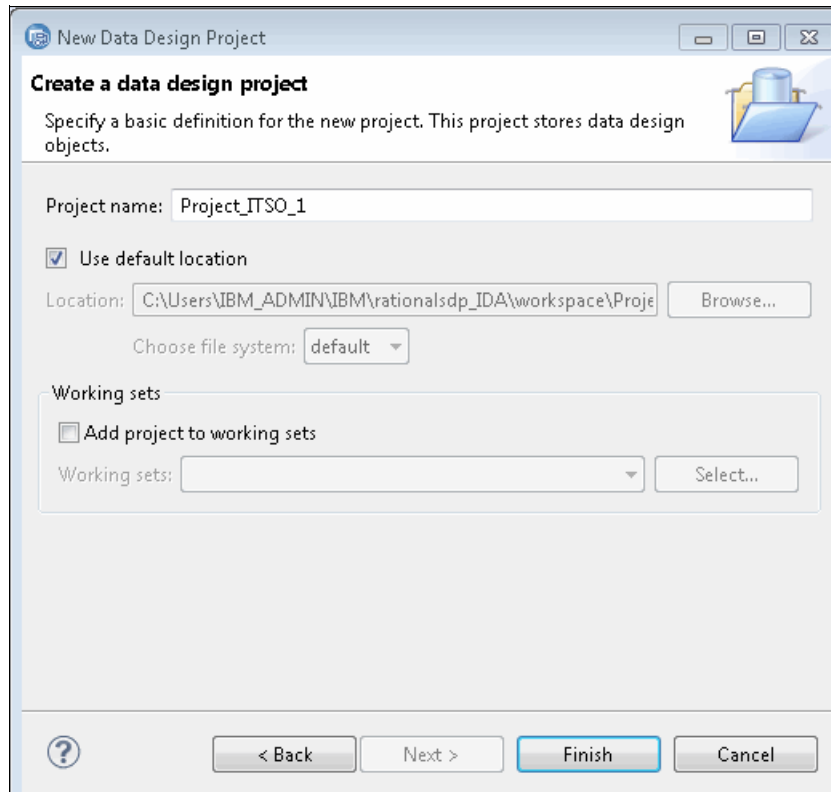


Figure 8-65 Specifying the Data Design Project name

4. You are returned to the main InfoSphere Data Architect interface. This window is the same one shown in Figure 8-62 on page 311. Now you may create a new connection to the database. Right-click **Database Connections** and select **New...**, as shown in Figure 8-66.

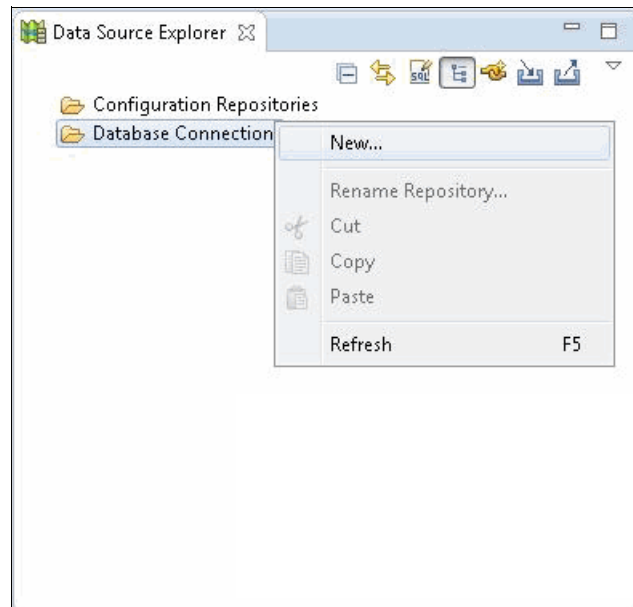


Figure 8-66 Creating a database connection

5. The New Connection window is shown in Figure 8-67. You need to set the database type to DB2 for i and set the following parameters for this connection:

- Host Name
- User Name
- Password

The default schema is an optional field.

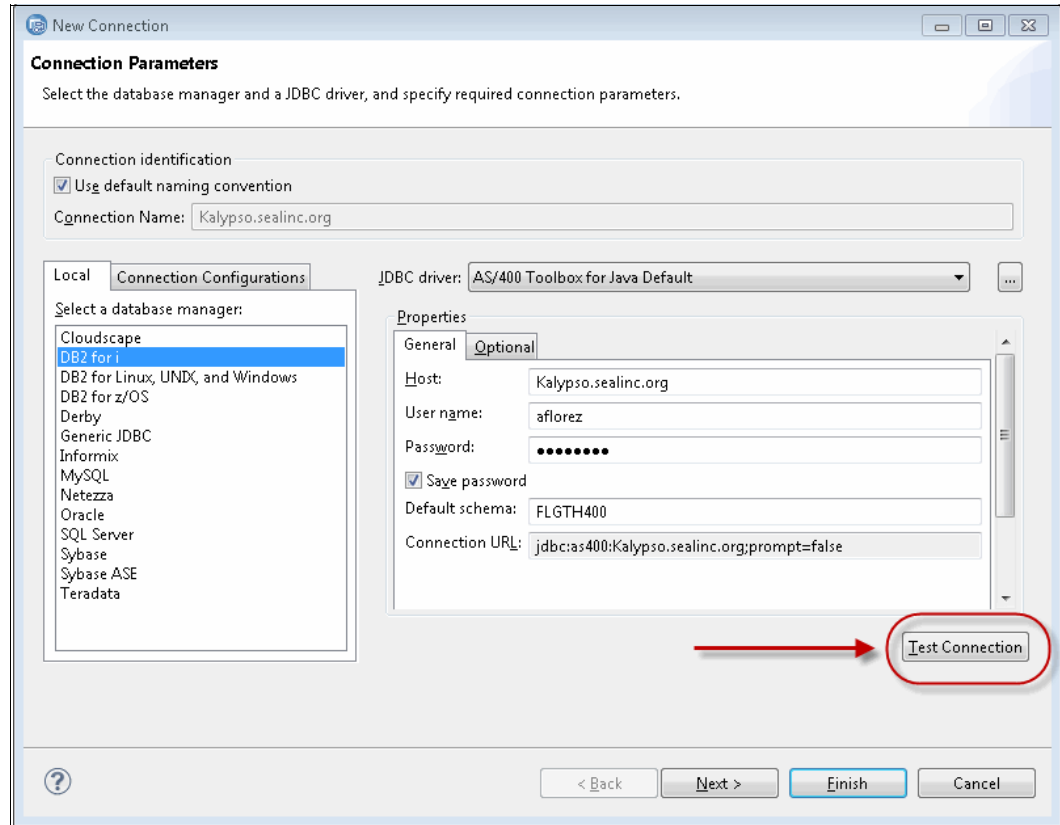


Figure 8-67 InfoSphere Data Architect - creating the database connection parameters

6. To ensure that you entered the correct information and things are working as expected, click **Test Connection**, as shown in Figure 8-67.

- When you know that the connection is working correctly, click **Next**. Specify the Physical data model, as shown in Figure 8-68. In this window, you can link this connection with a physical model. This field is optional and it can be left blank.

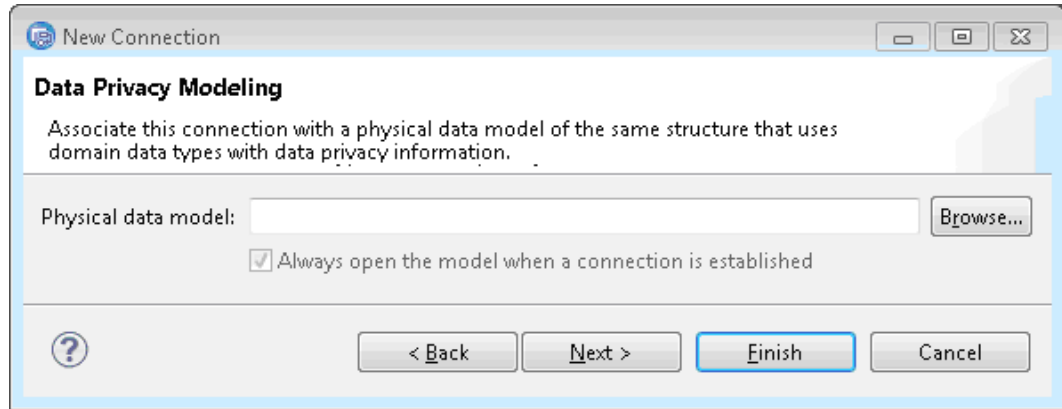


Figure 8-68 Physical data model for InfoSphere Data Architect

- Click **Next** to continue. Normally on IBM i there are many schemas. You can filter and see only some schemas. In this window, you can filter the schemas by expression or by selection. It is important to select **Disable filter** to enable the option filters, as shown in Figure 8-69.

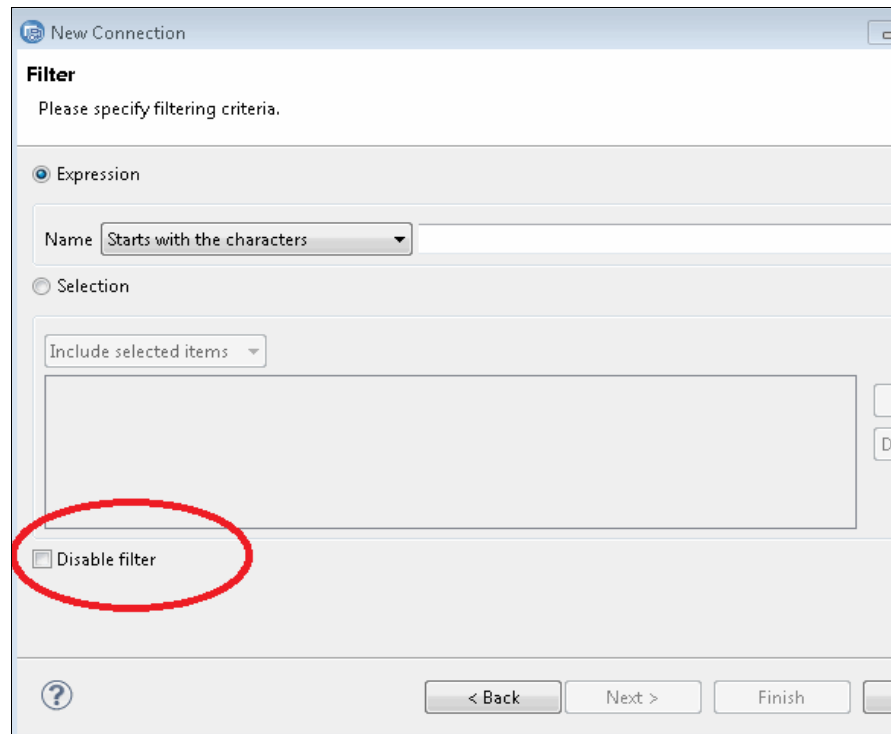


Figure 8-69 InfoSphere Data Architect Filter window

- You can filter by Expression, as shown in Figure 8-70 on page 317.

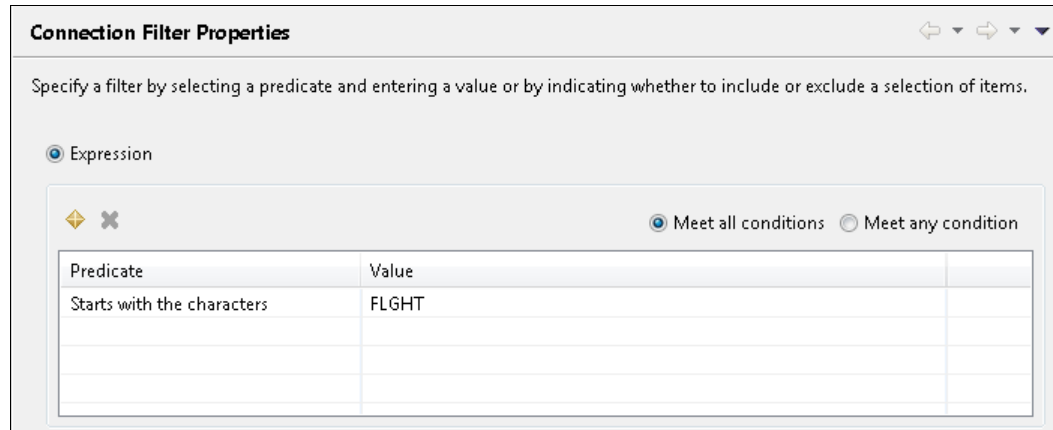


Figure 8-70 Filtering by expression

10. The other option is Filter by selection. You can select each schema to view it, as shown in Figure 8-71.

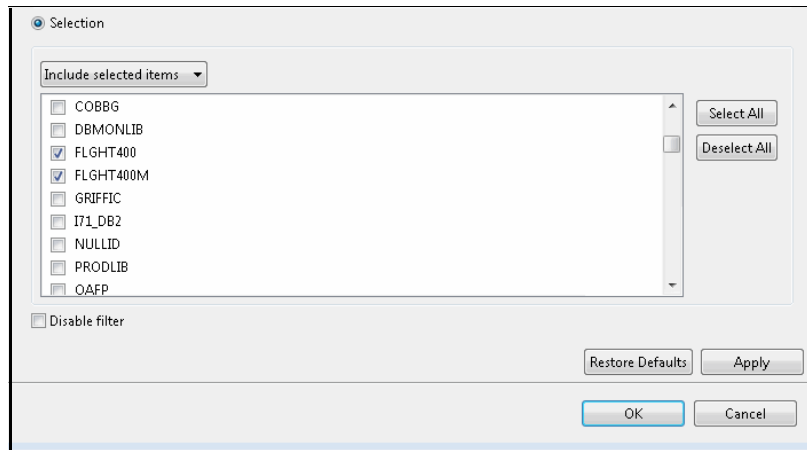


Figure 8-71 Filtering by selection

11. When you have selected the filters, click **Apply** and then **OK** to continue. You can see the connection that is defined in your workspace, as shown in Figure 8-72.

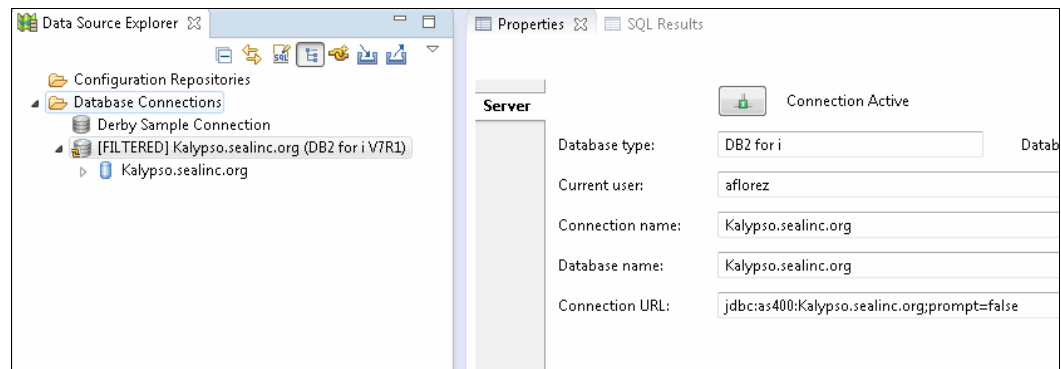


Figure 8-72 Connection created

12. You can work with DB2 for i from IBM InfoSphere Data Architect. If you look in the Data Source Explorer view, you can see the connection tree. You can expand each item, as shown in Figure 8-73.

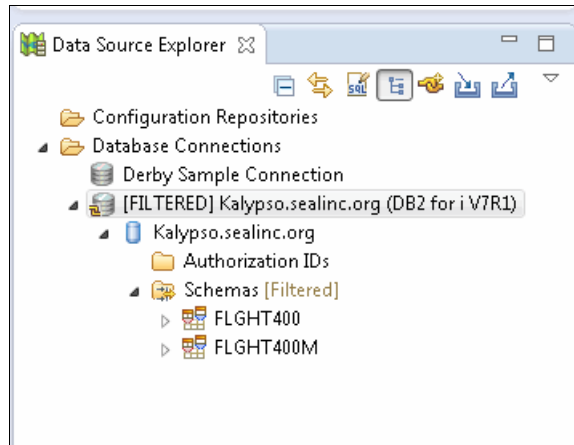


Figure 8-73 Connection that is created on InfoSphere Data Architect

Adding an existing model to the diagram

IBM InfoSphere Data Architect can diagram an existent model. To do so, complete the following steps:

1. Select the information to diagram, right-click **Tables**, and select **Add to Overview Diagram** **Diagram**, as shown in Figure 8-74.

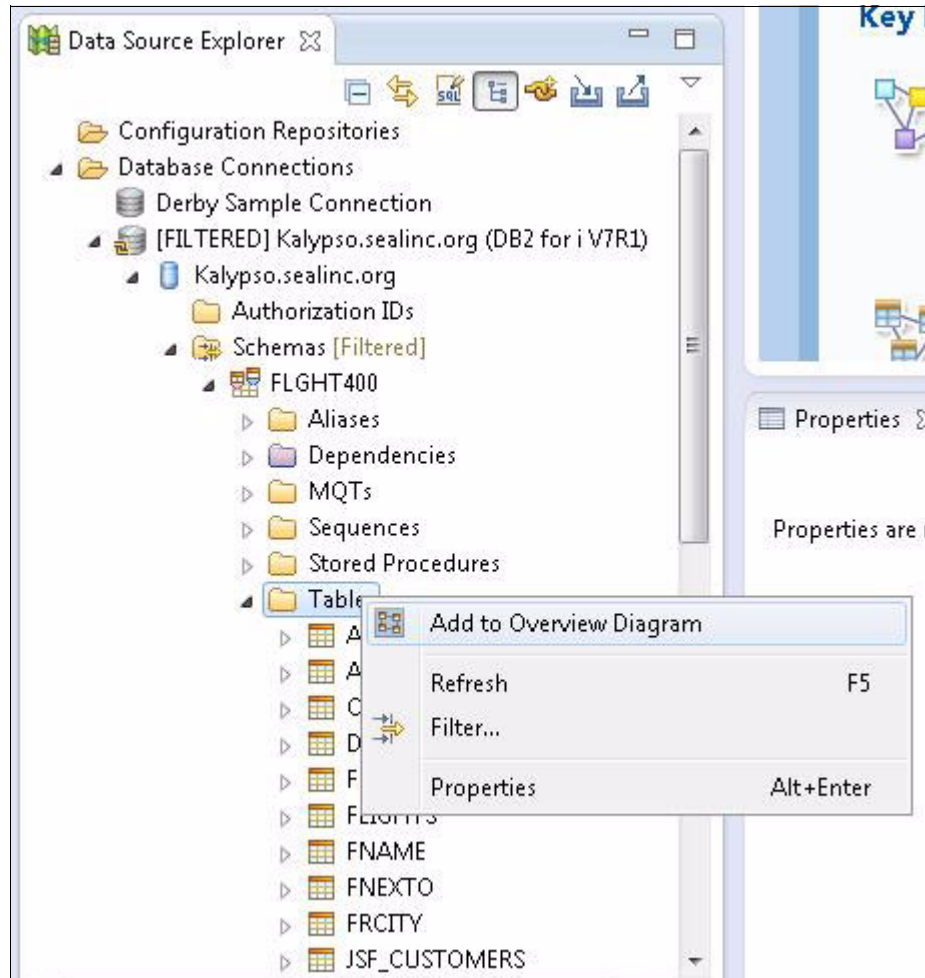


Figure 8-74 Adding to the Overview diagram

2. Select each table that you want to add to the diagram, as shown in Figure 8-75.

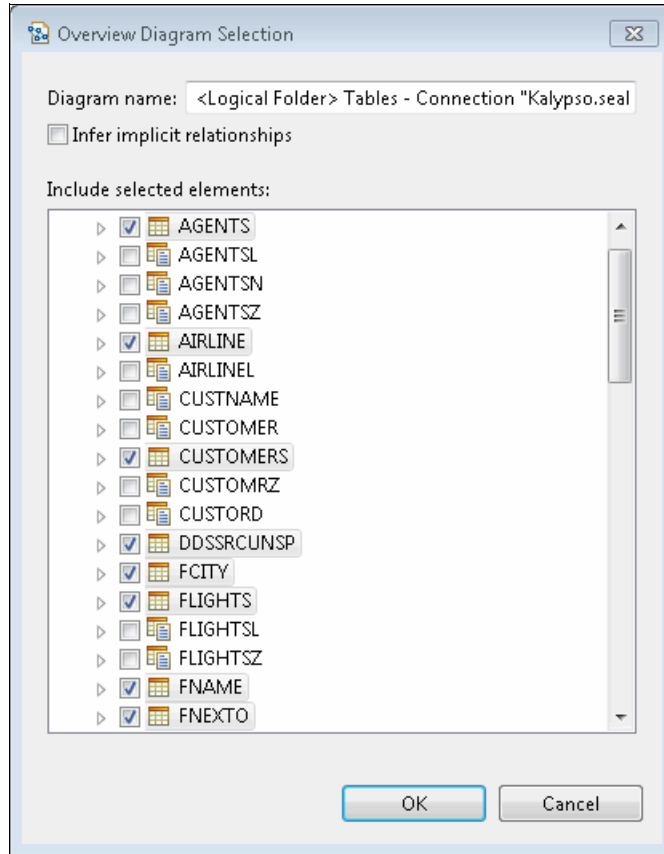


Figure 8-75 Selecting tables to diagram in InfoSphere Data Architect

Note: It is important not to select source physical files.

3. Click **OK** to continue. You see the diagram that is shown in Figure 8-76, which shows the diagram for the tables selected.

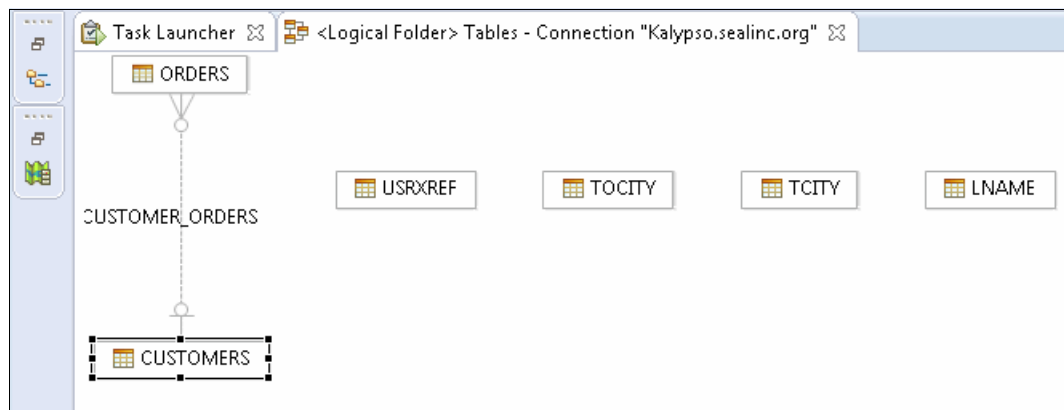


Figure 8-76 Logical Model Diagram on InfoSphere Data Architect

The diagram can be customized with the Personalize view of Logical Mode.

8.6.3 Managing DB2 for i with IBM i System Navigator

In this section, you learn how to manage DB2 for i with IBM i System Navigator. To do so, complete the following steps:

1. Start IBM i System Navigator by clicking **Start** → **All Programs** → **IBM i Access for Windows** → **System i Navigator**, as shown in Figure 8-77.

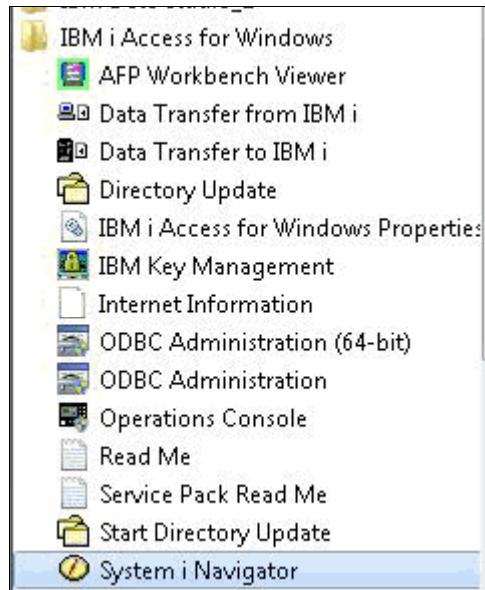


Figure 8-77 Starting IBM i Navigator

2. When you open the product, you see the main window, as shown in Figure 8-78.

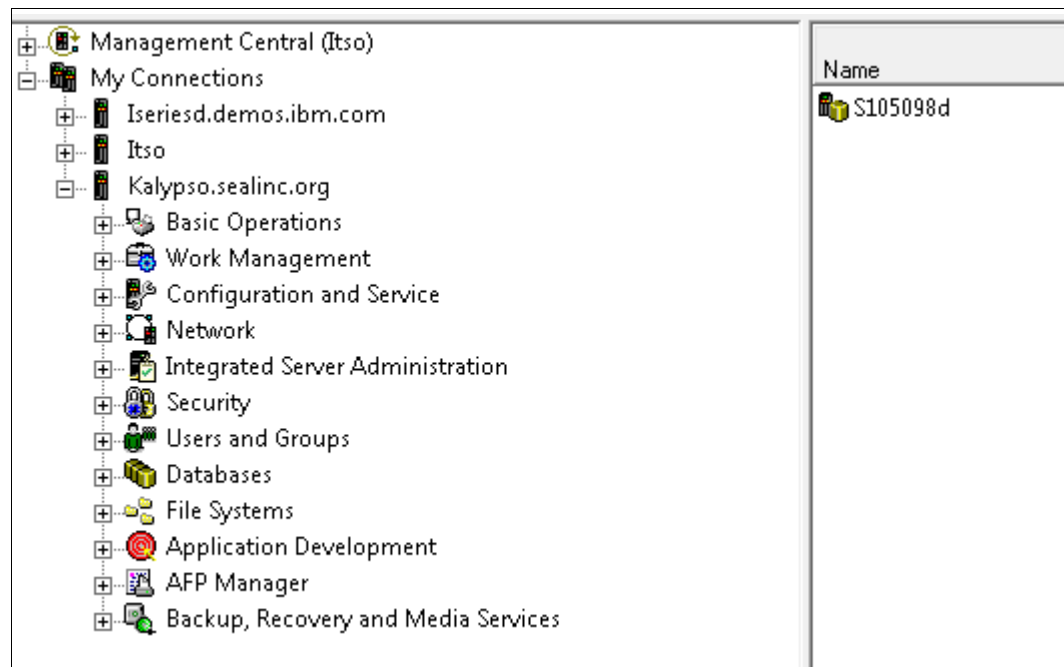


Figure 8-78 Main IBM i Navigator page

3. When you have specified a system to manage and have opened the navigation tree and signed on to the system, you are ready to start working with the database. Click the **Databases** option, as shown in Figure 8-79.

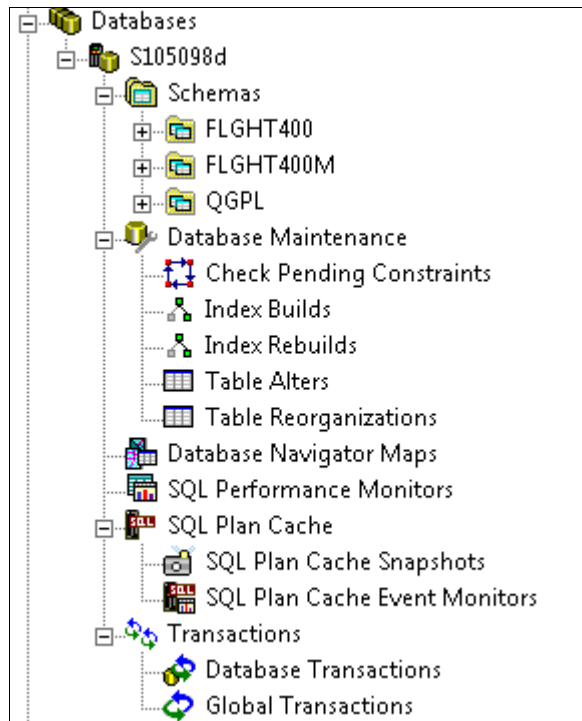


Figure 8-79 Menu options for the database container

The Database option contains many options that you can use. Here are a few of these options:

- Schemas: A schema is a database structure that contains your tables, views, and other object types. You can use schemas to group related objects and to find objects by name. Some tasks that you can perform with schemas are create a schema, select displayed schemas, remove a schema from the display list, and drop a schema.
 - Database Maintenance: Here you can find options for views to check pending constraints, index build, index rebuilds, alter tables, table reorganizations, and text index builds.
 - Database Navigator Maps: View a diagram of your database to see all the relationships in your database. The visual diagram that you create for your database is called a Database Navigator Map. In essence, the Database Navigator Map is a snapshot of your database and the relationships that exist between all of the objects in the map.
 - SQL Performance Monitor: The SQL Performance Monitor allows you to track the resources that SQL statements use. You can monitor specific resources or many resources. The information about resources that are used can help you determine whether your system and your SQL statements are performing as they should, or if they need some tuning. There are two types of monitors that you can choose for your resources: Summary SQL performance monitor and Detailed SQL performance monitor
4. In this example, we are going to show you how to view a Database Navigator Map. To get started, right-click the database instance name. In this case, it is S105098d, as shown in Figure 8-80 on page 323.

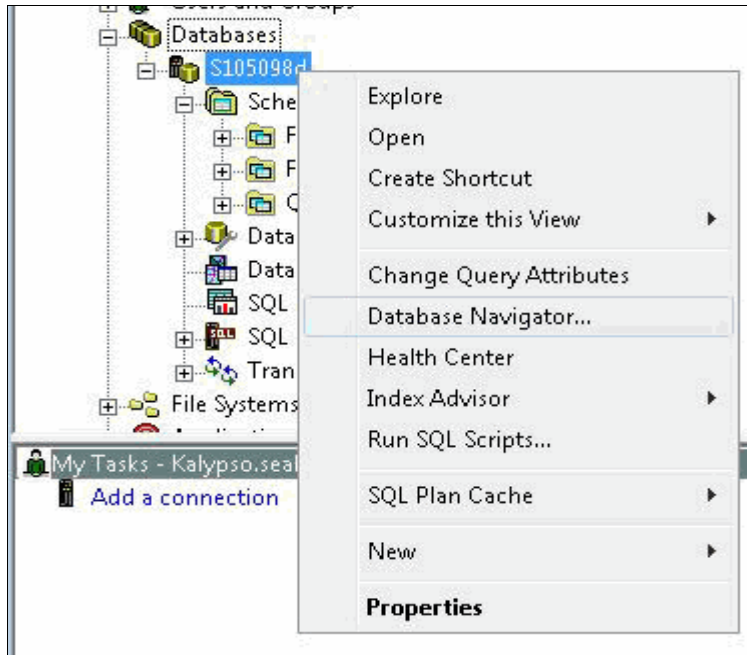


Figure 8-80 Right-clicking to start the Database Navigator support

- From the main Database Navigator window, you can search for objects or use the navigation tree to find what you are looking for, as shown in Figure 8-81.

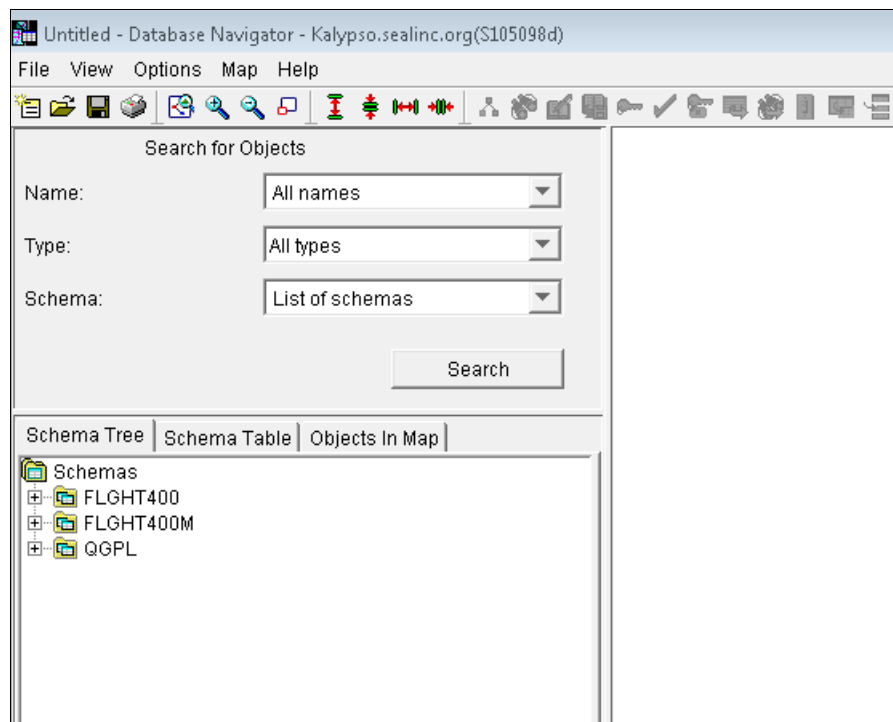


Figure 8-81 Database Navigator main page

6. To see the diagram map, open the FLIGHT400 database to find the tables view, right-click **Table**, and select **Add contents to Map**, as shown in Figure 8-82.

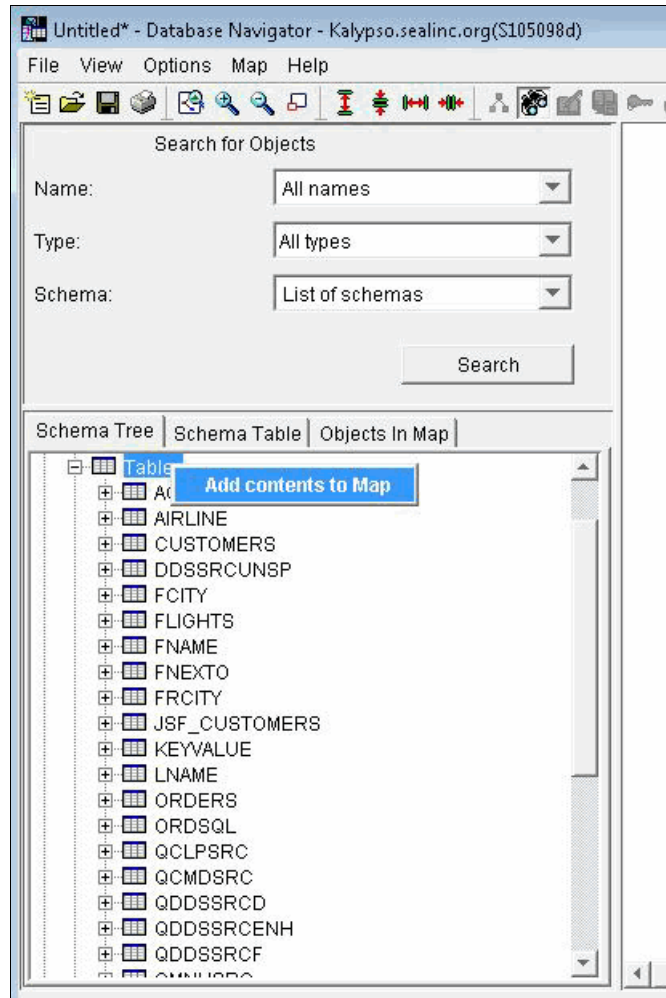


Figure 8-82 Opening the database and clicking the Add contents to Map option

7. You now see the navigator map for the selected objects, as shown in Figure 8-83.

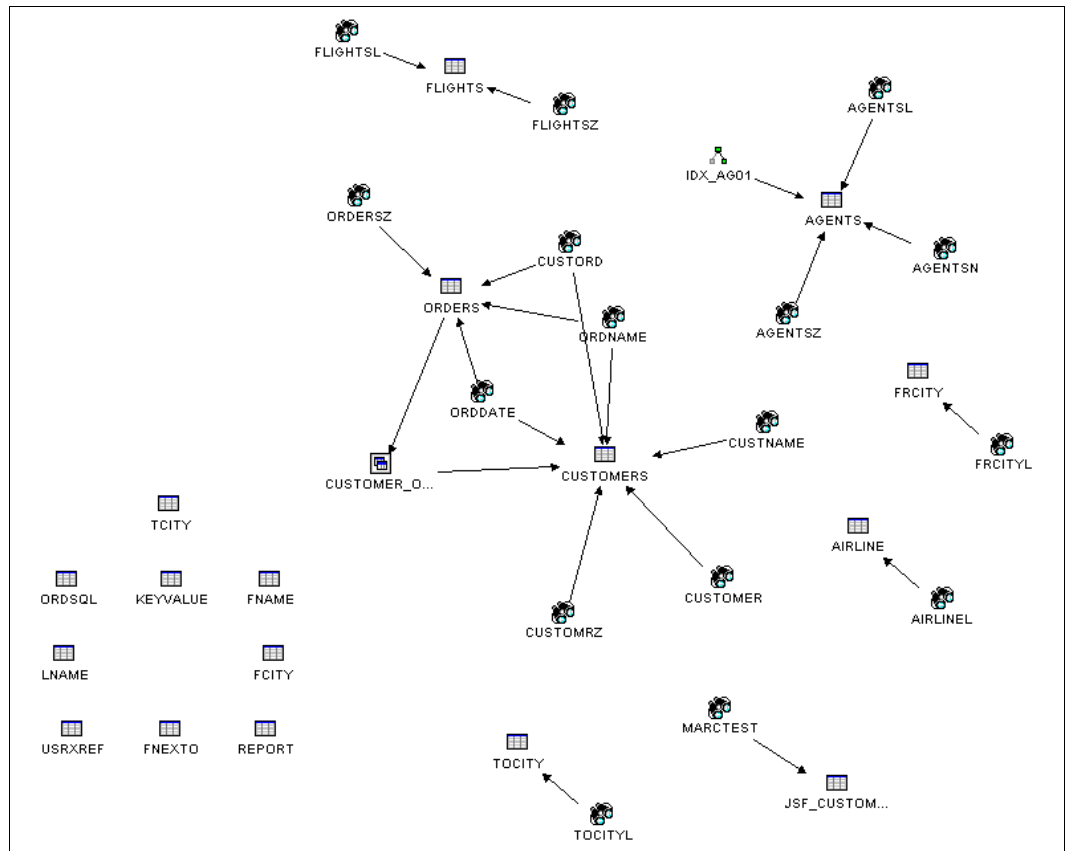


Figure 8-83 Map diagram view of the selected database objects

8. For large maps, there is an outline tool that allows you to zoom in on a selected section of the map. This enables you to see additional details. Click the **Outline** icon. Then, you can adjust what you need to look at. The icon is circled in the window that is shown in Figure 8-84.

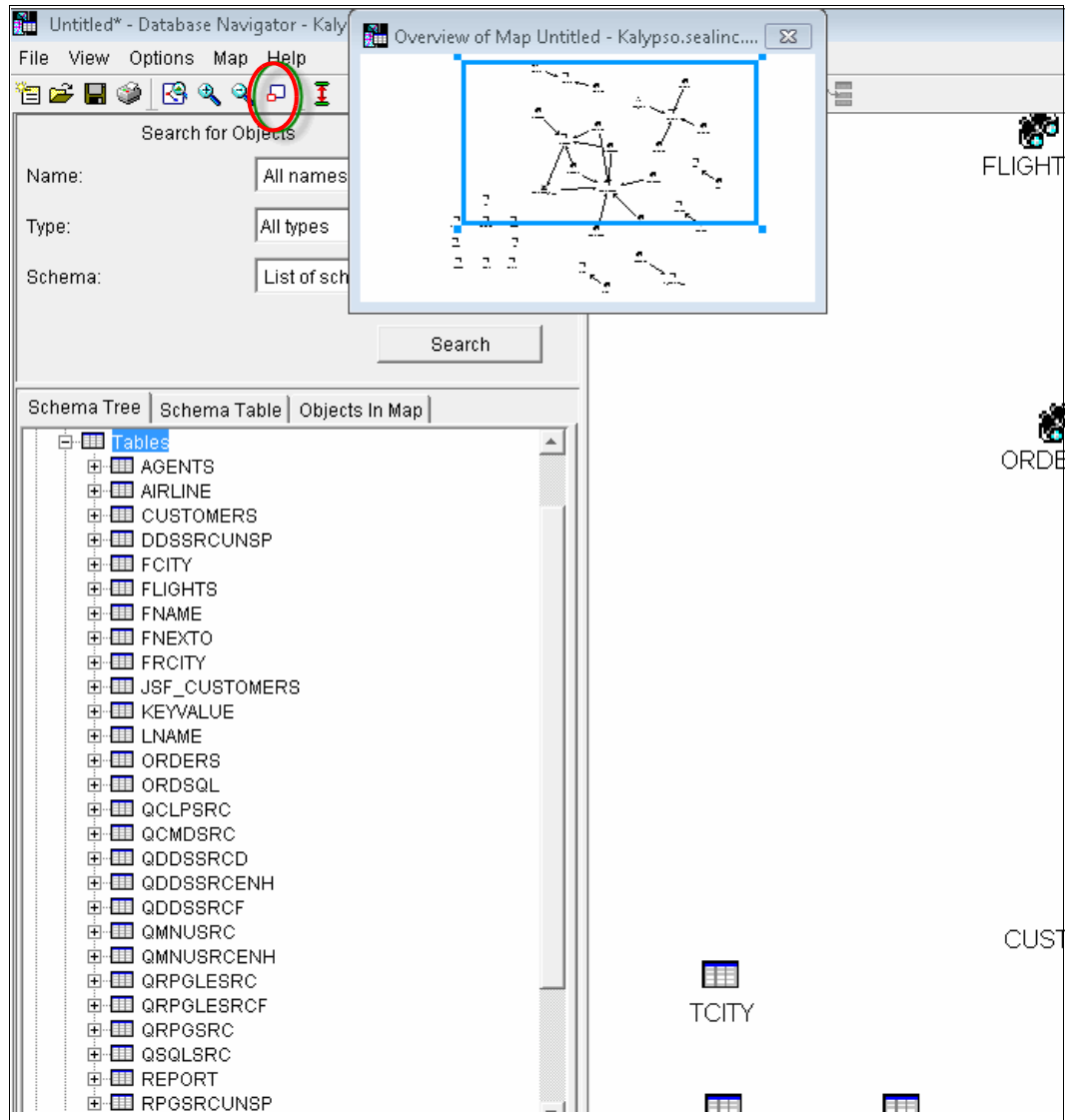


Figure 8-84 Using the Outline tool to narrow the view

9. In the Outline tool, you can select a specific area by creating a rectangle, as shown in Figure 8-85 on page 327. After selecting the area, you can see the detailed objects in the main window.

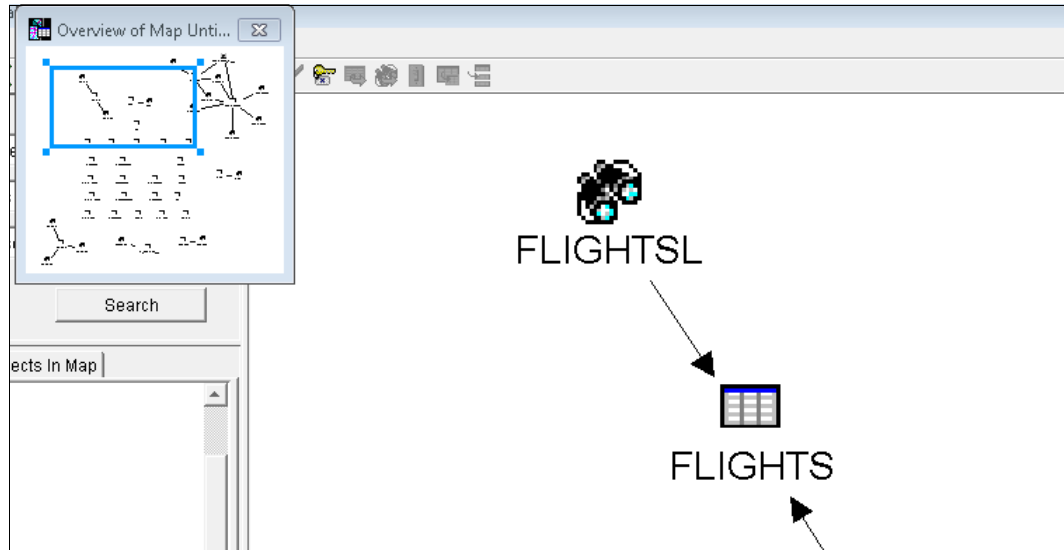


Figure 8-85 Zoomed in view using the Outline tool

- To make the map more focused on only the items you need, you can remove selected objects from the map. For example, in this case, we remove the source physical files, as shown in Figure 8-86.

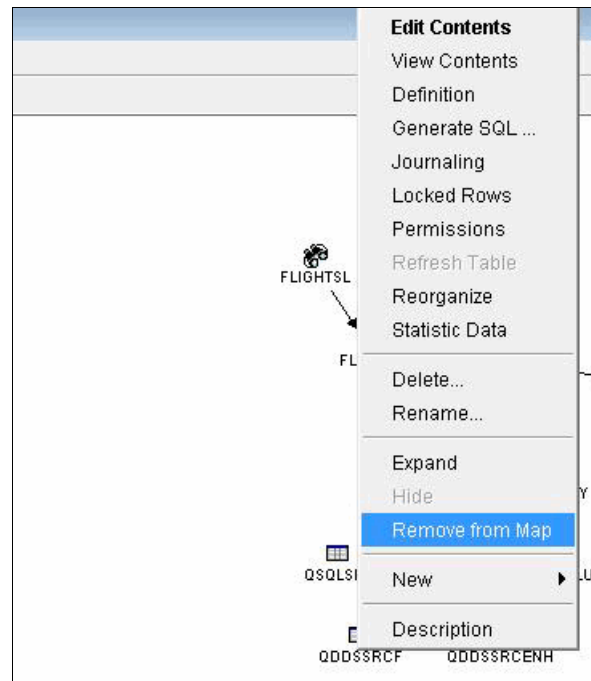


Figure 8-86 Outline view actions list

Note: If you select the **Remove from Map** option, the objects are removed from the map only, not the database. However, if you select the **Delete** option, the objects are deleted from the database.

There are many options that can be run against each object in the map. These depend on your authorization and the object type. Here are some of the options:

- ▶ Edit contents
- ▶ View contents
- ▶ Definition
- ▶ Generate SQL
- ▶ Journaling
- ▶ Looked rows
- ▶ Permissions
- ▶ Refresh table
- ▶ Reorganize
- ▶ Static Data
- ▶ Delete
- ▶ Rename
- ▶ Expand
- ▶ Hide
- ▶ Remove from map
- ▶ New (Alias, Index, Trigger)
- ▶ Description

For more information about using IBM i Navigator options, see the following resources:

- ▶ *Piloting DB2 for i5/OS with System i Navigator - V5R4* (downloadable lab), found at: <http://www.ibm.com/partnerworld/wps/servlet/ContentHandler/servers/enable/site/education/labs/e552/index.html>
- ▶ *Advanced Functions and Administration on DB2 Universal Database for iSeries*, SG24-4249
- ▶ *Stored Procedures, Triggers, and User-Defined Functions on DB2 Universal Database for iSeries*, SG24-6503

8.6.4 SYSTOOLS

SYSTOOLS is a set of DB2 for IBM i supplied examples and tools. SYSTOOLS is the name of a database supplied schema (library). SYSTOOLS differs from other DB2 for i supplied schemas (QSYS, QSYS2, SYSIBM, and SYSIBMADM) in that it is not part of the default system path. These tools are general-purpose tools and examples that are built by IBM.

It is the intention of IBM to add content dynamically to SYSTOOLS, either on base releases or through PTFs for field releases. A preferred practice for customers who are interested in such tools is to periodically review the contents of SYSTOOLS.

Using SYSTOOLS

You can generate the sample SQL procedures, learn how to call the procedures, and understand the outcome that is expected. You can also modify the procedure source to customize an example into your business operations.

Using the System i Navigator interface, click **Database**, expand the system, and click **Schemas**. Then, click the **SYSTOOLS** link. In the work area, right-click the top entry and select **Generate SQL**, as shown in Figure 8-87. This action uses the Generate Data Definition Language (QSQGDDDL) API to produce the **CREATE PROCEDURE (SQL)** statement. This statement is needed to create a replica of the IBM supplied procedure.

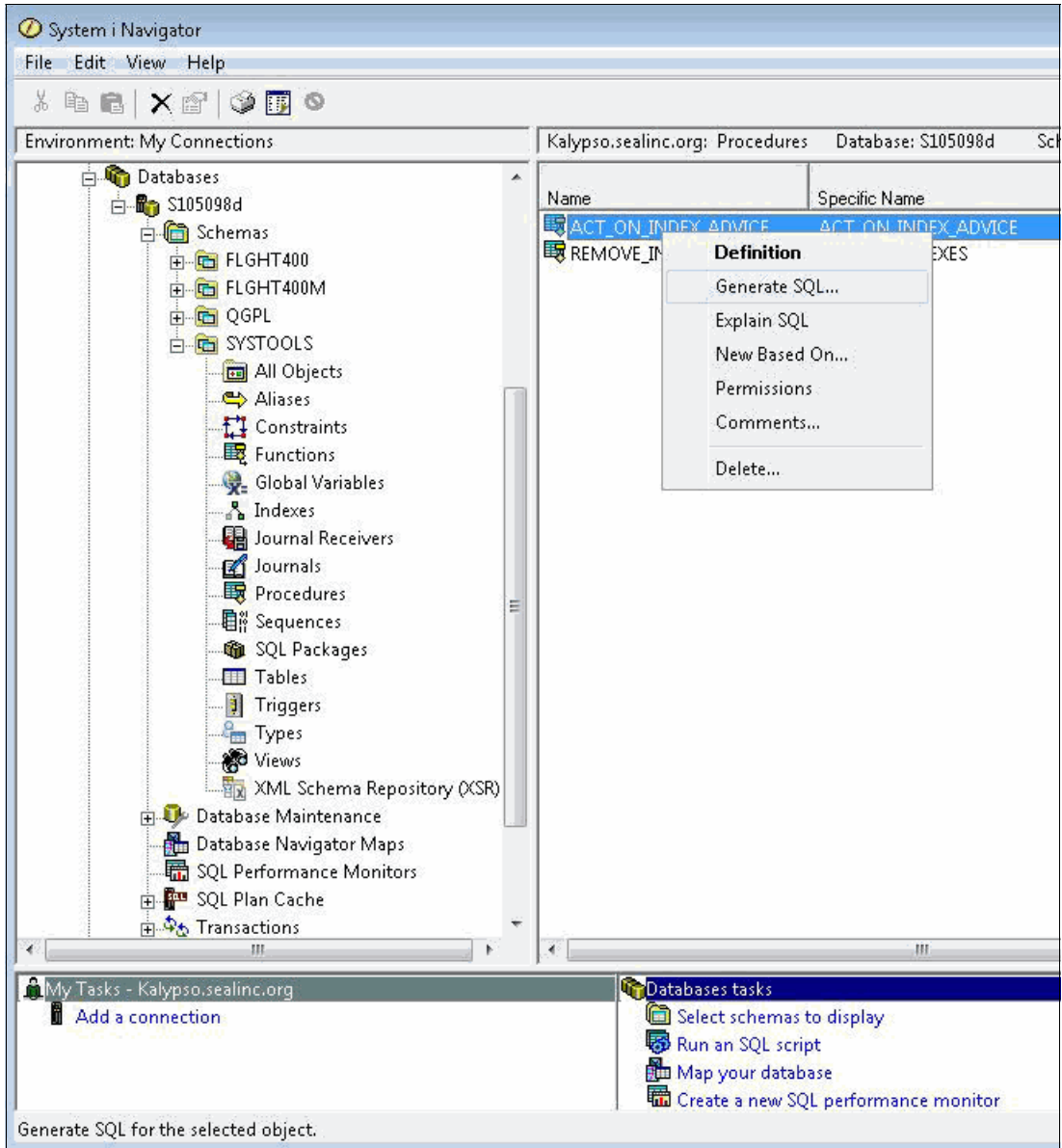
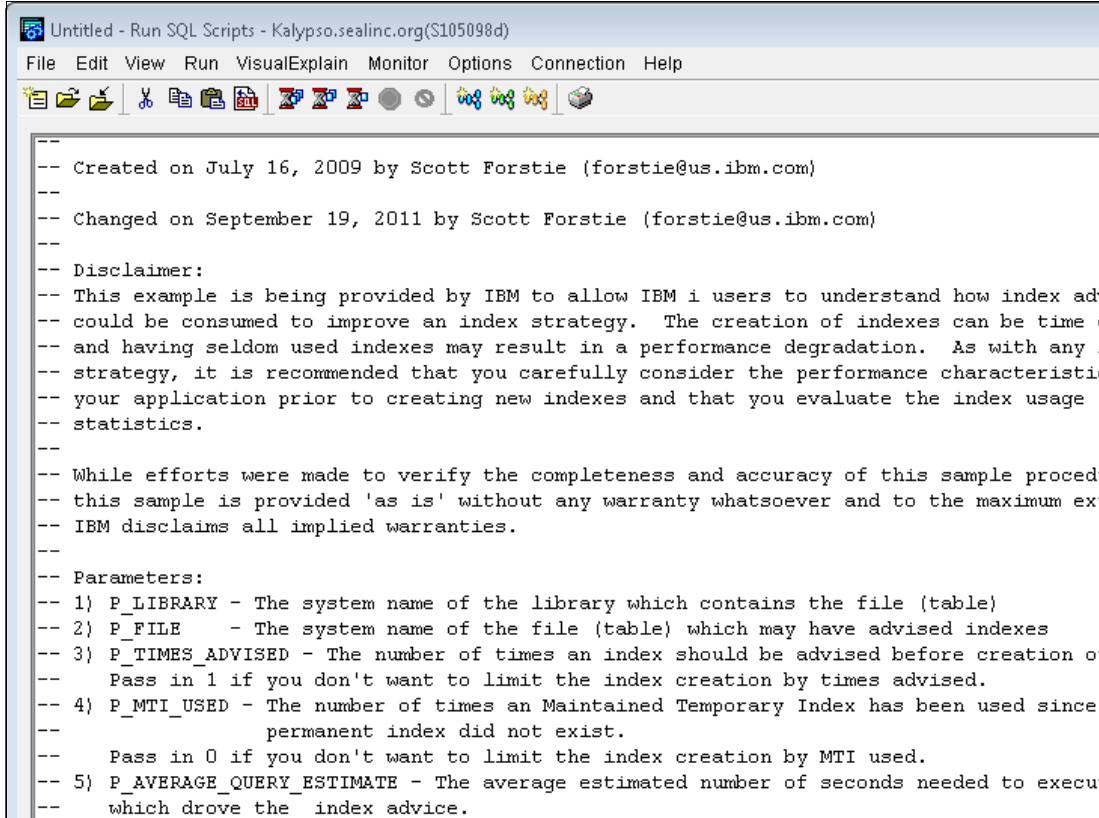


Figure 8-87 Selecting Generate SQL option for SYSTOOLS

After the Generate SQL action completes, The Run SQL Scripts window is active (as shown in Figure 8-88), which allows you to do the following tasks:

- ▶ Scroll down and read the procedure prolog.
- ▶ Understand how to call the procedure and the outcome that is expected.
- ▶ Modify the procedure source, including the procedure name and schema. This capability might be the most useful aspect of SYSTOOLS because it allows you to claim and customize quickly an IBM supplied example into your business operations.



```
---
-- Created on July 16, 2009 by Scott Forstie (forstie@us.ibm.com)
--
-- Changed on September 19, 2011 by Scott Forstie (forstie@us.ibm.com)
--
-- Disclaimer:
-- This example is being provided by IBM to allow IBM i users to understand how index ad
-- could be consumed to improve an index strategy. The creation of indexes can be time
-- and having seldom used indexes may result in a performance degradation. As with any
-- strategy, it is recommended that you carefully consider the performance characteristi
-- your application prior to creating new indexes and that you evaluate the index usage
-- statistics.
--
-- While efforts were made to verify the completeness and accuracy of this sample proced
-- this sample is provided 'as is' without any warranty whatsoever and to the maximum ex
-- IBM disclaims all implied warranties.
--
-- Parameters:
-- 1) P_LIBRARY - The system name of the library which contains the file (table)
-- 2) P_FILE - The system name of the file (table) which may have advised indexes
-- 3) P_TIMES ADVISED - The number of times an index should be advised before creation o
-- Pass in 1 if you don't want to limit the index creation by times advised.
-- 4) P_MTI_USED - The number of times an Maintained Temporary Index has been used since
-- permanent index did not exist.
-- Pass in 0 if you don't want to limit the index creation by MTI used.
-- 5) P_AVERAGE_QUERY_ESTIMATE - The average estimated number of seconds needed to execu
-- which drove the index advice.
```

Figure 8-88 Run QSQL window - reading, modifying, and rerunning the source

The IBM maintenance of SYSTOOLS includes periodically dropping and re-creating the IBM supplied objects. Customers are allowed to create their own objects within SYSTOOLS. However, if your user-created objects conflict with the IBM supplied objects, your objects might be deleted. The tools and examples in SYSTOOLS are considered ready for use. However, they are not subject to IBM Service and Support because they are not considered part of any IBM product.

SYSTOOL examples

This section looks at some procedures that were recently added to the SYSTOOLS schema. We focus on the following procedures:

- ▶ **CHECK_SYSROUTINE ()**
- ▶ **SYSTOOLS.CHECK_SYSCST**

These procedures were supplied with SF99701 Level 21 group PTF for IBM i 7.1. For anyone that is responsible for deployment and maintenance of multiple systems, the **CHECK_SYSRoutine** and **CHECK_SYSCST** procedures can help prevent much debugging or downtime. After procedure/function development, program/service program development, and save/restore operations, many customers want and expect that two systems contain identical catalog entries. These procedures provide a mechanism to confirm the catalog consistency and to understand quickly any differences.

SYSTOOLS.CHECK_SYSRoutine procedure details

These procedures allow you to automate disaster recovery preparedness checking. Given the complex nature of keeping SQL and external procedure/function database catalog entries in sync across systems, DB2 for i provides a catalog assessment utility that compares a target system's SYSRoutine catalog with the version on the local system:

```
CALL SYSTOOLS.CHECK_SYSRoutine (<target-database-name>,
<schema-to-compare>)
```

There are two parameters for the **CHECK_SYSTRoutine** function:

- ▶ **P_REMOTE_DB_NAME**: The name of remote database
- ▶ **P_SCHEMA_NAME**: The name of the schema to use as the comparison point

Here is an example of calling this procedure against another system:

```
CALL QGPL.CHECK_SYSRoutine('RCHAPTF3', 'SYSIBM');
```

The results are shown in Figure 8-89.

SERVER_NAME	SPECIFIC_SCHEMA	SPECIFIC_NAME	ROUTINE_SCHEMA	ROUTINE_NAME	ROUTINE_TYPE	ROUTINE_TYPE
RCHAPTF3	SYSIBM	USERS	SYSIBM	USERS	FUNCTION	SQL
RCHAPTF3	SYSIBM	SQLFUNCTIONS	SYSIBM	SQLFUNCTIONS	PROCEDURE	EXT
RCHAPTF3	SYSIBM	SQLFUNCTIONCOLS	SYSIBM	SQLFUNCTIONCOLS	PROCEDURE	EXT

Figure 8-89 Output for the **CHECK_SYSRoutine** call

SYSTOOLS.CHECK_SYSCST procedure details

Similar in concept to the **CHECK_SYSRoutine()** procedure, this procedure allows you to automate disaster recovery preparedness checking. Using the **CHECK_SySCST** procedure, you can compare constraints between a remote system and the local system:

```
CALL SYSTOOLS.CHECK_SYSCST (<target-database-name>,
<schema-to-compare>,
<optional-result-set-parameter>)
```

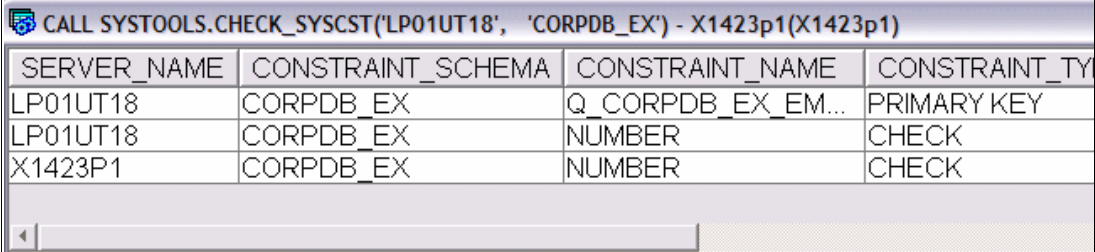
Here are the parameters for **CHECK_SYSCST**:

- ▶ **P_REMOTE_DB_NAME**: The name of remote database.
- ▶ **P_SCHEMA_NAME**: The name of the schema to use as the comparison point.
- ▶ **P_AVOID_RESULT_SET**: Optional parameter. Pass in the value 1 to avoid the result set.
 - If the **P_AVOID_RESULT_SET** parameter is not passed, the result set is returned to the caller.
 - If the **P_AVOID_RESULT_SET** parameter is specified and is *not* set to zero, no result set is returned and the caller can query the **SESSION.SYSCSTDIFF** table.

Here is an example of calling this procedure:

```
CALL SYSTOOLS.CHECK_SYSCST('LP01UT18', 'CORPDB_EX')
```

The results are shown in Figure 8-90.



SERVER_NAME	CONSTRAINT_SCHEMA	CONSTRAINT_NAME	CONSTRAINT_TYPE
LP01UT18	CORPDB_EX	Q_CORPDB_EX_EM...	PRIMARY KEY
LP01UT18	CORPDB_EX	NUMBER	CHECK
X1423P1	CORPDB_EX	NUMBER	CHECK

Figure 8-90 Output for the CHECK_SYSCST call

In this example, one constraint is missing on the master (local) database and a second constraint was disabled on the master.

SYSTOOLS.REMOVE_INDEXES procedure details

The **REMOVE_INDEXES** procedure accepts three parameters as criteria to guide the review of permanent index usage. The procedure does not consider all permanent indexes when reviewing index usage, but instead looks only at indexes that were created by using the naming scheme that is contained in the **SYSTOOLS.ACT_ON_INDEX_ADVICE** and **SYSTOOLS.HARVEST_INDEX_ADVICE** procedures.

Applications that use native database I/O (for example, RPG and COBOL) might have been coded to directly access keyed logical files, so you must avoid removing those types of database indexes.

Note: Even if you do not want to call the SYSTOOLS procedures, the SYSINDEXSTAT catalog example can be used to identify low-use indexes that are related to large, frequently changing tables.

Consider the **REMOVE_INDEXES** parameters:

- ▶ **P_LIBRARY - CHAR(10):** This is the system name of the library that contains the indexes that should be evaluated for pruning. If NULL is passed, the entire database is processed.
- ▶ **P_TIMES_USED - BIGINT:** If the number of times the index has been used by a query and used for statistics is less than this parameter, the index is considered underutilized. This input parameter is required.
- ▶ **P_INDEX_AGE - VARCHAR(100):** This parameter guides the procedure to evaluate indexes that have existed longer than a certain amount of time.

Similar to the **P_LIBRARY** parameter, this parameter limits the scope of indexes that are reviewed for acceptable usage. This parameter can be used as a labeled-duration expression in a query against the QSYS2/SYSINDEXSTAT statistical catalog. Any valid SQL labeled-duration expression can be used. This input parameter is required.

As shown in the following example, you can examine indexes within FLGHT400, removing indexes older than one month that have not been used:

```
CALL SYSTOOLS.REMOVE_INDEXES('FLGHT400', 1, ' 1 MONTH ')
```

You can also examine indexes from all schemas, removing indexes older than one week that have not been used at least 500 times:

```
CALL SYSTOOLS.REMOVE_INDEXES(NULL, 500, '7 DAYS ')
```

SYSTOOLS.HARVEST_INDEX_ADVICE procedure detail

This procedure is similar to the **SYSTOOLS.REMOVE_INDEXES** procedure. The difference is that this procedure places the **CREATE INDEX** statements in a target source physical file.

You can use it “as is” or customize it to fit your business logic and requirements. Here is its syntax:

```
CREATE PROCEDURE SYSTOOLS.HARVEST_INDEX_ADVICE (  
  IN P_LIBRARY CHAR(10) ,  
  IN P_FILE CHAR(10) ,  
  IN P_TIMES ADVISED BIGINT ,  
  IN P_MTI_USED BIGINT ,  
  IN P_AVERAGE_QUERY_ESTIMATE INTEGER ,  
  IN T_LIBRARY CHAR(10) ,  
  IN T_FILE CHAR(10) )
```

```
CL: DLTF FILE(QGPL/INDEXSRC);  
CL: CRTSRCPF FILE(QGPL/INDEXSRC) RCDLEN(10000);  
CALL SYSTOOLS.HARVEST_INDEX_ADVICE('CORPDB23', 'SALES', 100, 0, 0, 'QGPL',  
  'INDEXSRC');
```

HTTP functions added to SYSTOOLS

HTTP is the preferred way for communicating in resource-oriented architecture (ROA) and service-oriented architecture (SOA) environments. Use these RESTful services to integrate information sources that can be addressed through a URL and accessed with HTTP.

Here are the new DB2 for i HTTP functions and HTTP UDF names:

- ▶ **httpGetBlob**
- ▶ **httpGetClob**
- ▶ **httpPutBlob**
- ▶ **httpPutClob**
- ▶ **httpPostBlob**
- ▶ **httpPostClob**
- ▶ **httpDeleteBlob**
- ▶ **httpDeleteClob**
- ▶ **httpBlob**
- ▶ **httpClob**
- ▶ **httpHead**

8.6.5 XML

In Version 7.1, IBM enhanced DB2 for IBM i with a native XML data type, plus a number of functions and stored procedures for working with XML. The easiest way to understand the new function and how to use it is to take a close look at both the XML and relational models, and then consider the advantages that a native XML type within DB2 for i offers.

eXtensible Markup Language (XML) has become widely used within the last 10 years or so. XML has a wide number of potential uses, but one of special interest is the case where XML is used as a data model for transmitting information between different applications. Standards that are built on XML for transmitting data exist in every industry.

Relational databases (such as DB2 for i) have for many years been the method of choice for storing and retrieving business data. DB2 for i has always supported a relational data model for efficiently updating and querying information. IBM has an optimizer that knows how to find the fastest way to evaluate an SQL query, with minimal fine-tuning by a DB2 administrator.

Efficient storage, update, and query of XML data is a major challenge for the research community. The data model is designed for transmitting business data, and not so much with these performance ideals in mind.

In addition, query and data-manipulation languages that are written for XML typically do not yet include the Atomicity, Consistency, Isolation, and Durability (ACID) properties that are built into SQL. When using SQL and DB2 for i, commitment control and isolation levels ensure that even when multiple applications are accessing the same data in the database, the data always remains in a consistent state. IBM i is a robust platform, but if a catastrophic failure does occur, the database is in a reliable state when the system is brought back up again.

The bottom line is that business data is often stored in relational databases, and this is going to be the case for the foreseeable future. However, the same business data often must be represented in an XML model so that it can be transmitted between the database and an XML-aware application.

The result is that XML data and relational data are habitually forced into an unlikely partnership. By supporting a native XML type in Version 7.1, DB2 for i adds the capability for XML and relational data to coexist within the same database.

Sample XML document

Although XML data is transmitted in a text format, it is incorrect to think of the XML data type as a character data type. Consider a serialized XML document for a car purchase order and see how an XML document describes both data and relationships between data within the document:

```
<?xml version="1.0" encoding="UTF-8"?>
<vo:Order
  xmlns:vo="http://www.example.ibm.com/VehicleOrder"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://www.example.ibm.com/VehicleOrder VehicleOrder.xsd">
  <vo:Customer xmlns="http://www.example.ibm.com/Customer">
    <Name>
      <GivenNames>
        <GivenName> Jane </GivenName>
        <GivenName> Smith </GivenName>
      </GivenNames>
      <FamilyName> Doe </FamilyName>
    </Name>
    <Email>username@example.com</Email>
  </vo:Customer>
  <vo:Vehicle xmlns="http://www.example.com/Vehicle">
    <Year>1931</Year>
    <VIN>1234ABCDEFG</VIN>
    <Price>8700000.00</Price>
    <Description>A very expensive car</Description>
  </vo:Vehicle>
  <vo:Timestamp>2012-01-30T13:01:00-06:00</vo:Timestamp>
</vo:Order>
```

This document indicates that it conforms to an XML schema (VehicleOrder.xsd) that describes the data model and should be used for validation. Schema design is a complex topic that is well outside the scope of this section; however, a basic overview is necessary to understand the XML data model.

Overview of an XML schema

Although having a schema is not required by DB2 for i or the W3C standard, there are many cases where it is necessary for practical purposes. When transmitting data between different applications, a formal schema that describes the allowed structure and data types of the data being transferred is usually a prerequisite.

When provided, the XML schema defines the structure, data types, and valid values in the XML document. XML schemas conform to a W3C standard that supports built-in primitive types, complex and simple user-defined types, and also types that are derived by either extension or restriction. Because XML itself is used as the schema definition language, a simple text editor can be used to define the XML schema. However, because of the language's complexity, tools such as Rational Application Developer are widely available and employed to define the schema's source code.

The schema for this example is shown (with IBM Rational Application Developer) in Figure 8-91.

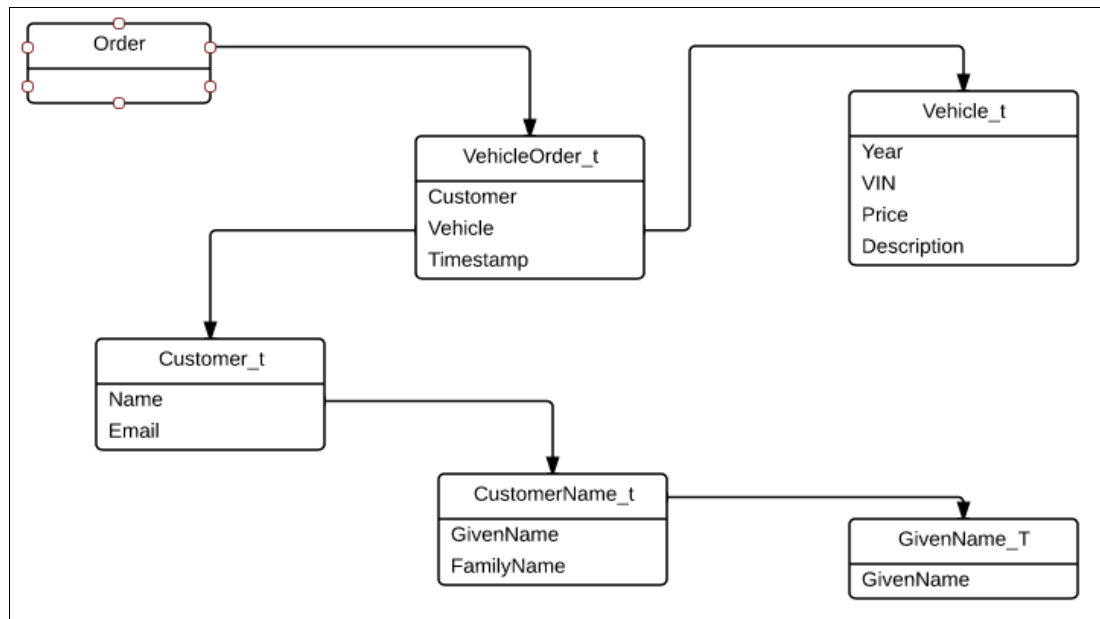


Figure 8-91 Schema definition

The schema and sample document shows that XML documents represent much more than simple character data. In this example, each leaf element has an XML primitive type that is associated with it. Each primitive type has both a well-defined textual representation and a well-defined set of valid values. The result is that a parser and application can always convert the character representation of the data to a value of the specified type. For example, the `vo:Timestamp` element's value can be converted from character '2012-01-30T13:01:00-06:00' to an XML `dateTime` value that represents 01:01 pm on January 30, 2012 in a time zone six hours behind Coordinated Universal Time.

If the document is valid according to the schema, `vo:TimeStamp` *must* contain a valid representation of an XML `dateTime` value. Because of this, an application knows that it can always cast the text in the element to a `dateTime` and perform date and time calculations or comparisons using the value.

The schema also defines the cardinality of the elements in the document. `Customer` and `Vehicle` allow one-or-many occurrences (1..*) and `Timestamp` is defined to have an exactly one (1..1) occurrence requirement.

Even in cases where a schema is required by the application, validation of documents against the schema can often be an optional step. If you trust that an XML document is constructed in accordance with the schema, then there is no need to perform validation. DB2 for i supports validation of an XML document against a schema using the `XMLVALIDATE` built-in function, but such validation is not required and is often avoided for performance reasons.

Hierarchical and ordered relationships

Hierarchical and ordered relationships that occur in an XML document are difficult to represent in a relational model. Shredding an XML document into relational tables usually requires a deep understanding of both the relationships in the data and the application's needs to be effective; this design is seldom a trivial process. This dilemma is a common reason for wanting to store XML data as a native XML type in DB2 for i.

Consider this example:

```
<Name>
  <GivenNames>
    <GivenName> John </GivenName>
    <GivenName> Doe </GivenName>
  </GivenNames>
  <FamilyName> Smith </FamilyName>
</Name>
```

To find out the value of “Name”, simply concatenate the values of all of the contained elements (in document order). The result is the expected result of “John Doe Smith”.

Using this same model, you can retrieve the values of the more specific components. In this case, FamilyName has a value of “Smith”.

It is common in XML for elements to have an occurrence requirement of one-or-many, such as GivenName in this example. Obviously, some individuals have more parts to their names than others, and the number of GivenName elements vary from document to document. Some cultures do not use family names, and so the FamilyName element is optional; it does not exist in some documents.

Although the order of tuples in a relational database is normally not significant, order is important in an XML document. In this document, the first GivenName clearly comes before the second GivenName. Both GivenName elements clearly come before the FamilyName element. It is good to allow this ordering to vary; customers from cultures where the family name is written first can represent their name correctly. The application can determine the correct value of Name and yet still correctly retrieve the individual components when necessary.

Even in this trivial example, shredding the document into relational tables and still retaining the hierarchical and order relationships that exist in the XML document is not easy. A native XML type allows XML to be stored in a model that preserves all of these relationships.

Schema evolution

Another important property of XML is that the structure of documents tends to rapidly evolve as business requirements change. In the vehicle order example, suppose that the business wants to track more specific information for each type of vehicle that it sells. XML schemas allow types to be extended. This means that you can easily change the schema so that subclass types are defined for each specific type of vehicle. The previously drawn schema can be changed to include two new extensions to Vehicle, one for cars and one for trucks, as shown in Figure 8-92 on page 337. An XML document can now include a “Vehicle” element that is a “Car_t” or a “Truck_t”.

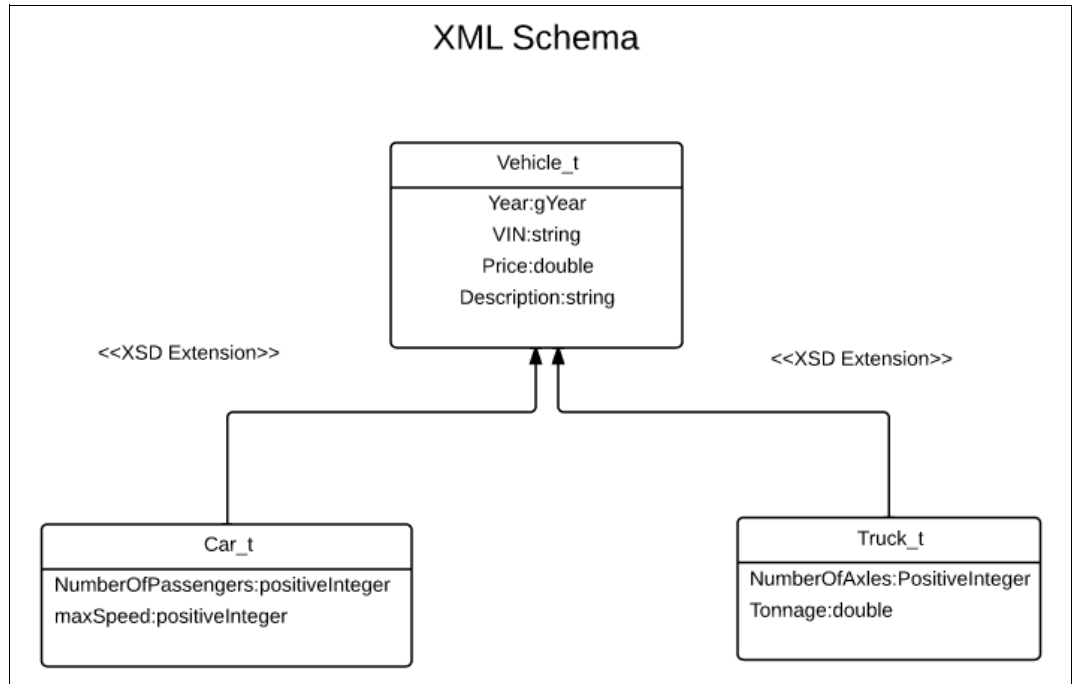


Figure 8-92 XML schema

Using this extension to the schema, the Vehicle element for a car might look like this:

```

<vo:Vehicle xmlns=http://www.example.ibm.com/Vehicle
  xsi:type="Car_t">
  <Year>1931</Year>
  <VIN>1234ABCDEFG</VIN>
  <Price>8700000.00</Price>
  <Description>A very expensive car </Description>
  <NumberOfPassengers>2</NumberOfPassengers>
  <MaxSpeed>200</MaxSpeed>
</vo:Vehicle>
  
```

A vehicle element for a truck has a different structure:

```

<vo:Vehicle xmlns="http://www.example.ibm.com/Vehicle"
  xsi:type="Truck_t">
  <Year>2012</Year>
  <VIN>9999ABCDEFG</VIN>
  <Price>50000.00</Price>
  <Description> A big Red Fire Truck </Description>
  <NumberOfAxles>2</NumberOfAxles>
  <Tonnage> 8 </Tonnage>
</vo:Vehicle>
  
```

The structure of future XML documents continues to become progressively more variable as new types of vehicles are added, which can be expected to occur regularly. Because of type extension, the schema remains compatible with earlier versions, and the structures of existing documents do not change as new vehicle types are added.

The cardinality of an element can also evolve in an XML model. To go from allowing only a single occurrence of an element to allowing a one-or-many occurrence is easy. First, the schema is updated, and then in cases where the higher cardinality applies, more elements are added in the document. The new schema remains compatible with earlier versions of existing documents.

Relational models depend on a rigid structural definition and often take much more work to evolve with the needs of the business. In general, creating a subclass or one-to-many relationship in a relational model results in new tables and foreign key constraints being added to the database. For some applications, hundreds of tables and complex queries are required to model XML data as relational. Therefore, the relational model tends not to evolve as fast or as easily as its XML counterpart.

This capability of XML to evolve with the business is another strong motivator for keeping XML data in its native form within DB2 for i.

XMLTABLE

The **XMLTABLE** built-in table function can be used to retrieve the contents of an XML document as a result set that can be referenced in SQL queries, as shown in Figure 8-93.

The addition of **XMLTABLE** support to DB2 for i users makes it easier for data centers to balance and extract value from a hybrid data model where XML data and relational data coexist.

Assume that you have created a table with an XML column (Figure 8-93):

```
create table emp (doc XML);
```

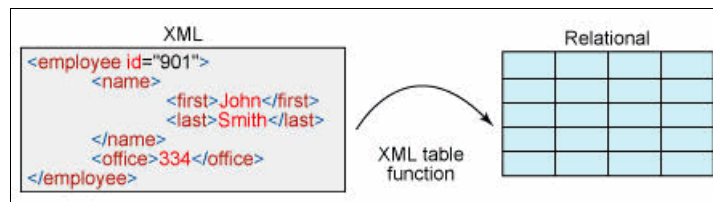


Figure 8-93 XML to relational

The table contains two rows that look like this:

► Row #1:

```
<dept bldg="101">
  <employee id="901">
    <name>
      <first>Jane</first>
      <last>Smith</last>
    </name>
    <office>344</office>
    <salary currency="USD">55000</salary>
  </employee>
  <employee id="902">
    <name>
      <first>John</first>
      <last>Doe</last>
    </name>
    <office>216</office>
    <phone>555-555-0123</phone>
  </employee>
```



```

    </dept>
► Row #2:
    <dept bldg="114">
      <employee id="903">
        <name>
          <first>Jane</first>
          <last>Doe</last>
        </name>
        <office>415</office>
        <phone>555-555-0124</phone>
        <phone>555-555-0125</phone>
        <salary currency="USD">64000</salary>
      </employee>
    </dept>

```

In the **XMLTABLE** function invocation, you specify a row-generating XPath expression, and in the columns clause, one or more column-generating expressions.

In this example, the row-generating expression is the XPath expression `$d/dept/employee`. The passing clause refers to the XML column doc of the table `emp`:

```

SELECT X.*
FROM emp,
XMLTABLE ('$d/dept/employee' passing emp.doc as "d"
  COLUMNS
  EMPID      INTEGER      PATH '@id',
  FIRSTNAME  VARCHAR(20)   PATH 'name/first',
  LASTNAME   VARCHAR(25)   PATH 'name/last') AS X

```

The row-generating expression is applied to each XML document in the doc column and produces one or multiple employee elements (subtrees) per document.

The output of the **XMLTABLE** function contains one row for each employee element. Hence, the output that is produced by the row-generating XPath expression determines the cardinality of the **SELECT** statement.

The **COLUMNS** clause is used to transform XML data into relational data. Each of the entries in this clause defines a column with a column name and an SQL data type. In the examples above, the returned rows have three columns that are named `EMPID`, `FIRSTNAME`, and `LASTNAME`, with data types `Integer`, `Varchar(20)`, and `Varchar(25)`. The values for each column are extracted (by using the XPath expression) from the employee elements, which are produced by the row-generating XPath expression.

The extracted values are cast to the SQL data types. For example, the path `name/first` is applied to each employee element to obtain the value for the `FIRSTNAME` column. The row-generating expression provides the context for the column-generating expressions. You can typically append a column-generating expression to the row-generating expression to get an idea of what a given **XMLTABLE** function returns for a column.

The result of the previous query is shown in Table 8-2.

Table 8-2 XMLTABLE example

EMPID	FIRSTNAME	LASTNAME
901	Jane	Smith
902	John	Doe
903	Jane	Doe

The SQL/XML support that is included in DB2 for i 7.1 offers a built-in and standardized solution for working with XML from within SQL. XMLTABLE adds significant value by allowing SQL queries to query both XML and relational data in the same query. Working XML data within a relational model is often times a non-trivial task. XMLTABLE offers a high degree of flexibility by supporting a wide range of XPath step expressions, predicates, and built-in functions that can be used within the row and column generating expressions.

A general overview for how to use XMLTABLE to reference XML content is available in the *IBM i Database SQL XML Programming Guide*, found at:

<http://pic.dhe.ibm.com/infocenter/iseriess/v7r1m0/topic/rzasp/rzasppdf.pdf>

For the complete syntax of XMLTABLE function, see the XMLTABLE function topic in the *DB2 for i SQL Reference*, found at:

<http://pic.dhe.ibm.com/infocenter/iseriess/v7r1m0/index.jsp?topic=%2Fdb2%2Frbafzscaxmltable.htm>

8.6.6 Using RPG to use DB2 XML support

This section presents several examples of using RPG to use the SQL/XML function that is delivered as part of IBM DB2 for i 7.1. It shows how to embed SQL statements into an RPG program to perform relational queries that involve XML data and produce XML results. The SQL/XML publishing functions and the recently announced XMLTABLE function are demonstrated later in this section.

Modernizing and web-enabling applications are some of the important goals for any business, and a challenge for enterprises that have been around for a while. These companies usually want to focus their IT efforts on improving the infrastructure that has brought them success in the past, instead of writing new applications. On the IBM i platform, modernizing frequently means web-enabling a database application that is written in Integrated Language Environment (ILE) RPG. Because of the many XML standards that exist for transmitting data over the web, having XML capabilities that are available in RPG is important. What an RPG programmer must realize is that when DB2 for i introduced a native XML data type, every language that supports embedded SQL (including RPG) received a whole set of new options for dealing with XML alongside traditional database data.

Using data from an XML document in a relational query

In the FLGHT400 schema, you have a FLIGHTS file, which has information about every flight; FLGHT400 is a relational database table. Another program or interface provides an XML document to the program requesting that a report is generated for the flight that has a matching flight number. An example request document is shown in Figure 8-94 on page 341.

Flight #	City Deap.	City	Deaperture	Day	Time Deaperture
1200002	ABY	Albany		Monday	09:03 AM

Figure 8-94 Example report

The first RPG program retrieves an XML information request from a file and uses it to produce a report in a spool file. You could imagine that you are reading this data from standard input, or from a socket, but for a simple example, a stream file is enough to get the idea.

Implementing the query in SQL allows the DB2 optimizer to determine the best way to retrieve the relational data, and revise that determination if another access plan that can perform better becomes available later. This makes it highly desirable to use SQL as much as possible to retrieve data from the database.

A less-than-satisfying solution that has been done before is to extract the values from the XML document into host variables by using RPG opcodes, convert the values into an SQL appropriate type, and finally use an SQL query to retrieve the relational records. Special consideration of type conversions is necessary with this approach because XML data types often do not share lexical representation as their SQL equivalents. For example, the data type for `xs:dateTime` does not have the same lexical format as the SQL time stamp data type, nor does it match a native RPG time stamp type.

In this example, the `XML-INTO` or `XML-SAX` opcodes can be used to extract the flight number in to a host variable, followed by some data type conversion processing, and (finally) an SQL query to retrieve the matching records from the database table. This solution involves significant processing outside of the DB2 awareness, and introduces a greater risk of error.

Another option that has been used in the past is to use a transformation function to transform an XML document into an SQL query string. The query string is later prepared and run. This is often a complex process that has the same problem as the previous solution. It also has the additional concern that a malicious user might be able to construct an XML document that causes an unintended SQL statement to run.

With the new XML support built into DB2, the XML handling can be included in an embedded SQL query that returns the requested relational data. The embedded query handles the data type conversions and ensures that the data that is obtained from the XML document is used only as values in the query. For this case, the SQL is shown in Figure 8-95.

```

SELECT FLIGHT_NUMBER, DEPARTURE_INITIALS, DEPARTURE, DAY_OF_WEEK,
       ARRIVAL_INITIALS, ARRIVAL, DEPARTURE_TIME, ARRIVAL_TIME,
       AIRLINES, SEATS_AVAILABLE, TICKET_PRICE, MILEAGE
FROM   FLGHT400.FLIGHTS,
       XMLTABLE('FlightInfoRequest'
               PASSING XMLPARSE(DOCUMENT GET_XML_FILE(:file_name))
               COLUMNS "FLGNB" INTEGER ) info_req
WHERE  FLIGHTS.FLIGHT_NUMBER = INFO_REQ."FLGNB";

```

Figure 8-95 SQL query for program 1

The `GET_XML_FILE` function returns a BLOB from the stream file at the specified path. The XML data in the BLOB has been converted to UTF-8 encoding. The function is a handy way to get XML data into the database from an IFS path and, at the same time, avoids the need to copy the XML data from the file into a host variable.

The **XMLPARSE** function is used to convert the BLOB data into an instance of the XML data.

In the query that is shown in Example 8-25, **XMLTABLE** returns a result set with one row for each flight, having the column **FLGNB** from the XML document that has been retrieved from the file. Because the result set from **XMLTABLE** is joined with the database table, you can use this query to fetch the records in which you are interested.

The complete listing of the program is included in Example 8-25.

Example 8-25 RPG program to report flight information

```
Ctl-opt ALWNULL(*USRCTL) DEBUG(*YES) MAIN(PRINTRPT)
      ACTGRP(*NEW);
      // Output File, we don't need an input file
      Dcl-f FLIGHT_RPT Printer;

      //*****
      // These Variables are used to hold the values for
      // each column of the Result Rows
      //*****
      Dcl-s ORDERID Packed(9);
      Dcl-s ORDER_TS Timestamp;
      Dcl-s PRICE Packed(9:2);
      Dcl-s PRODUCT Char(25);
      Dcl-s CUST_Email Char(25);

      Dcl-s FLGNBR Packed(9);
      Dcl-s DEPARIN Char(3);
      Dcl-s DEPARTURE Char(16);
      Dcl-s DAY Char(16);
      Dcl-s ARRIVIN Char(3);
      Dcl-s ARRIVAL Char(16);
      Dcl-s DEPARTIM Char(16);
      Dcl-s ARRIVTIM Char(16);
      Dcl-s AIRLINES Char(16);
      Dcl-s SEATS Packed(9);
      Dcl-s TICKET Packed(9);
      Dcl-s MILEAGE Packed(9);

      //*****
      // The XML Document is read from this file on disk
      //*****
      Dcl-s file_name Char(25) Inz('/home/ajfloresz/flight.xml')

      //*****
      // These variables are used to hold information about
      // an SQL error if an error should occur.
      //*****
      Dcl-s SQLMESSAGE Varchar(3200);
      Dcl-s SQLMESSAGEL Char(107);
      Dcl-s SQLMESSAGEI Int(10) Inz(0);
      Dcl-s RETSCODE Packed(5);

      Dcl-pr PRINTRPT EXTPGM('PRINTRPT') End-pr;
```

```

//*****
//*****
// PROCEDURE IMPLEMENTATION
//*****
//*****

Dcl-proc PPRINTRPT;

//
// Need commitment control for working with LOB locators
//
Exec SQL
    Set Option commit=*CHG;

// declare the cursor
// This cursor is the key for everything, it handles both the parsing
// of the input XML document, the extraction of the data into columns,
// and the conversion of the columns to the correct SQL data types.
exec sql DECLARE C1 CURSOR FOR
    SELECT FLIGHT_NUMBER, DEPARTURE_INITIALS, DEPARTURE, DAY_OF_WEEK,
           ARRIVAL_INITIALS, ARRIVAL, DEPARTURE_TIME, ARRIVAL_TIME,
           AIRLINES, SEATS_AVAILABLE, TICKET_PRICE, MILEAGE
    FROM FLGHT400.FLIGHTS,
         XMLTABLE('FlightInfoRequest'
                  PASSING XMLPARSE(DOCUMENT GET_XML_FILE(:file_name))
                  COLUMNS "FLGNB" INTEGER ) info_req
    WHERE
        FLIGHTS.FLIGHT_NUMBER = INFO_REQ."FLGNB";
//open cursor
exec sql OPEN C1;

// output spool file headings
Write RTit;

// fetch first row
exec sql FETCH C1 INTO :FLGNBR, :DEPARIN, :DEPARTURE, :DAY,
                       :ARRIVIN, :ARRIVAL, :DEPARTIM, :ARRIVTIM, :AIRLINES,
                       :SEATS, :TICKET, :MILEAGE;

//exec sql FETCH C1 INTO :ORDERID, :ORDER_TS, :PRODUCT, :PRICE,
//                       :Cust_Email;

// for each successful fetch
DOW SQLCOD = 0;
// output details
Write RData;

// fetch next row
exec sql FETCH C1 INTO :FLGNBR, :DEPARIN, :DEPARTURE, :DAY,
                       :ARRIVIN, :ARRIVAL, :DEPARTIM, :ARRIVTIM, :AIRLINES,
                       :SEATS, :TICKET, :MILEAGE;
//exec sql
//FETCH C1 INTO :ORDERID, :ORDER_TS, :PRODUCT, :PRICE,

```

```

//      :Cust_Email;
ENDDO;
////////////////////////////////////
// check for error
////////////////////////////////////
IF SQLCOD <> 100 ; // 100 is no more records, which is what we expect
  exec sql GET DIAGNOSTICS CONDITION 1
    :SQLMESSAGE = MESSAGE_TEXT,
    :RETSCODE = DB2_RETURNED_SQLCODE;
// dump sqlcode and message into report
  Write RData;
// Print the SQL Message text in lines of 100 bytes
DOW SQLMESSAGEI < %len(SQLMESSAGE);
  IF %len(SQLMESSAGE) - SQLMESSAGEI > %len(SQLMESSAGEL);
    SQLMESSAGEL = %subst(SQLMESSAGE:SQLMESSAGEI+1:%len(SQLMESSAGEL));
  ELSE;
    SQLMESSAGEL = %subst(SQLMESSAGE:SQLMESSAGEI+1);
  ENDIF;
  MESSAGEL = SQLMESSAGEL;
  SQLMESSAGEI = SQLMESSAGEI + %len(SQLMESSAGEL);
  MESSAGEI = SQLMESSAGEI;
  Write RError;
ENDDO;
endif;
////////////////////////////////////
// close cursor and spool file
////////////////////////////////////
exec sql close C1;
CLOSE(E) *ALL;
RETURN;

```

End-proc;

The output for this program is the report that is shown in Figure 8-94 on page 341.

Creating an XML response from relational data

A second scenario must be considered. Suppose that you have the same type of information request, but this time you want to send the requester an XML response, instead of generating a spool file report from the query.

The trick here is to use your XML publishing functions to construct the XML response document. The simplest method is to use a common table expression to build the inner parts of the XML document, and then construct the outer layers of the document by arrogating the inner pieces.

The SQL query that is used in this second program is shown in Figure 8-96 on page 345.

```

WITH matching_flight as (
  SELECT
    XMLELEMENT(NAME "MatchingFlight",
      XMLFOREST(
        FLIGHT_NUMBER AS "FLIGHT_NUMBER",
        DEPARTURE_INITIALS AS "DEPARTURE_INITIALS" ,
        DEPARTURE AS "DEPARTURE",
        DAY_OF_WEEK AS "DAY_OF_WEEK" ,
        ARRIVAL_INITIALS AS "ARRIVAL_INITIALS" ,
        ARRIVAL AS "ARRIVAL",
        DEPARTURE_TIME AS "DEPARTURE_TIME",
        ARRIVAL_TIME AS "ARRIVAL_TIME",
        AIRLINES AS "AIRLINES",
        SEATS_AVAILABLE AS "SEATS_AVAILABLE",
        TICKET_PRICE AS "TICKET_PRICE",
        MILEAGE AS "MILEAGE")
      ) AS FLIGHT
  FROM FLIGHT400.FLIGHTS,
  XMLTABLE('FlightInfoRequest'
    PASSING XMLPARSE(DOCUMENT GET_XML_FILE(:file_name))
    COLUMNS "FLGNB" INTEGER
  ) flight_req
  WHERE
    FLIGHTS.FLIGHT_NUMBER = flight_req."FLGNB"
  )
  SELECT
    XMLSERIALIZE(
      XMLDOCUMENT(
        XMLELEMENT(NAME "InfoRequestResponse",
          XMLAGG(matching_flight.flight)
        )
      ) AS VARCHAR(15000) CCSID 37 INCLUDING XMLDECLARATION
    ) AS RESPONSE
  INTO :ResultXML
  FROM matching_flight;

```

Figure 8-96 SQL Query for program 2

For the request in Figure 8-96, the RPG program looks like Example 8-26 on page 346.

Using the publishing functions to construct an XML document makes it much easier to create a well-formed XML document from relational data. The data types are automatically converted from SQL to XML. The constructor functions always ensure that each XML tag is well-formed. DB2 manages the encoding of the XML document and makes sure that the encoding declaration is correct when the document is serialized.

Example 8-26 on page 346 contains the complete RPG example. It includes the code to load the information request from a stream file, perform the query, and write out an XML response to an output stream file. We can easily imagine how the example might be extended to perform the input and output using sockets or HTTP connections instead of files.

You should be aware that Figure 8-96 has been formatted for readability. As the indenting and formatting of an XML document is not required by the standard or by a consumer, DB2 does not add line breaks or indentation to the document during serialization. If you want to see an XML file that is formatted for display, you can easily open it in a web browser or an XML editor. When no style sheet is available, most web browsers display the XML as a document tree.

This particular program encodes the XML response in CCSID 37 (EBCDIC). An actual web application is more likely to use UTF-8, but 37 is easier to work with from green-screen interfaces, and it is a trivial change to alter the program to work with CCSID 1208 instead.

A limitation in this example program is that the result document cannot be greater than about 15 KB in size. If truncation or any other SQL problem occurs, the sample program constructs an XML document that includes the SQL error information and uses that as a response. A real program can have better handling routines.

The method that is used to create a response for errors is interesting. Some developers might find the approach of using embedded SQL to create an XML document from host variables to be simpler to write than a traditional string concatenation solution because it is easier to ensure that the XML document remains well-formed.

Example 8-26 contains the completed program.

Example 8-26 RPG program to create the XML

```
Ctl-opt ALWNULL(*USRCTL) DEBUG(*YES) MAIN(SENDRPTX)
      DFTACTGRP(*NO) ACTGRP(*NEW);
//*****
// The XML Document is read from this file on disk
//*****

Dcl-s file_name Char(35)
      Inz('/home/ajflorez/flight_input_doc.xml')
//*****
// The XML Response is written to this file on disk
//*****

Dcl-s resp_file Char(37)
      INZ('/home/ajflorez/flight_output_doc.xml');
//*****
// Result XML Value
//*****
// This has the limitation of only supporting 15000 bytes.
// We could use an SQL BLOB to support up to 2G, but for this example
// 15K is more than enough.
//*****
Dcl-s ResultXML Varchar(15000);
//*****
// These variables are used to hold information about an SQL error if
// an error should occur.
//*****
Dcl-s SQLMESSAGE Varchar(3200);
Dcl-s RETSCODE Packed(5)          S          5P 0
Dcl-s fd Int(10);
Dcl-s rc Int(10);
Dcl-pr SENDRPTX EXTPGM('SENRPTX') End-pr;
Dcl-pr open Int(10) Extproc(*Dclcase);
      path Pointer value options(*string);
      oflag Int(10) value;
      mode Uns(10) value options(*nopass);
      codepage Uns(10) value options(*nopass);
End-pr;
// Definition of IFS Write Procedure
Dcl-pr write Int(10) extproc(*Dclcase);
      filedes Int(10) value;
      buf Pointer value;
      nbyte Uns(10) value;
```



```

End-pr;
// Definition of IFS Close Procedure
Dcl-pr close Int(10) extproc(*Dclcase);
  filesdes Int(10) value;
End-pr;

//*****
//*****
// PROCEDURE IMPLEMENTATION
//*****
//*****
Dcl-proc SENDRPTX;
//
// Need commitment control for working with LOB locators
//
Exec SQL
  Set Option commit=*CHG;
exec sql declare :ResultXml VARIABLE CCSID 37;
exec sql declare :SQLMESSAGE VARIABLE CCSID 37;
// This query is the key for everything, it handles both the parsing
// of the input XML document, the extraction of the data into columns,
// the conversion of the columns to the correct SQL data types for
// the join, in addition, this query generates the XML result
exec sql
  WITH matching_flight as (
    SELECT
      XMLELEMENT(NAME "MatchingFlight",
        XMLFOREST(
          FLIGHT_NUMBER AS "FLIGHT_NUMBER",
          DEPARTURE_INITIALS AS "DEPARTURE_INITIALS" ,
          DEPARTURE AS "DEPARTURE",
          DAY_OF_WEEK AS "DAY_OF_WEEK" ,
          ARRIVAL_INITIALS AS "ARRIVAL_INITIALS" ,
          ARRIVAL AS "ARRIVAL",
          DEPARTURE_TIME AS "DEPARTURE_TIME",
          ARRIVAL_TIME AS "ARRIVAL_TIME",
          AIRLINES AS "AIRLINES",
          SEATS_AVAILABLE AS "SEATS_AVAILABLE",
          TICKET_PRICE AS "TICKET_PRICE",
          MILEAGE AS "MILEAGE")
        ) AS FLIGHT
    FROM FLGHT400.FLIGHTS,
      XMLTABLE('FlightInfoRequest'
        PASSING XMLPARSE(DOCUMENT GET_XML_FILE(:file_name))
        Columns "FLGNB" INTEGER
          ) flight_req
    WHERE
      FLIGHTS.FLIGHT_NUMBER = Flight_Req."FLGNB"
  )
  SELECT
    XMLSERIALIZE(
      XMLDOCUMENT(
        XMLELEMENT(NAME "InfoRequestResponse",
          XMLAGG(matching_flight.flight)
        )

```

```

        ) AS VARCHAR(15000) CCSID 37 INCLUDING XMLDECLARATION
    ) AS RESPONSE
    INTO :ResultXML
    FROM matching_flight;

////////////////////////////////////
// check for error
// If error, build invalid request response
////////////////////////////////////
IF SQLCOD <> 0 ;
    exec sql GET DIAGNOSTICS CONDITION 1 :SQLMESSAGE = MESSAGE_TEXT,
        :RETSCODE = DB2_RETURNED_SQLCODE;

    exec sql VALUES
        XMLSERIALIZE(
        XMLDOCUMENT(
        XMLELEMENT(NAME "SQLError",
            XMLELEMENT(NAME "SQLCODE",
                :RETSCODE),
            XMLELEMENT(NAME "MESSAGE",
                :SQLMESSAGE)
        )
        )
        AS VARCHAR(15000) CCSID 37
    ) INTO :ResultXML;
ENDIF;
////////////////////////////////////
// Write response into a stream file
////////////////////////////////////
EVAL fd = open(resp_file: 74 : 511);
EVAL rc = write(fd:%addr(ResultXML)+2:%len(ResultXML));
EVAL rc = close(fd);
RETURN;
End-proc;

```

You have seen two examples where embedded SQL has been included into an RPG program to perform database activities that involve both the relational and XML data models. You have seen how to include XML data in an SQL query, and how to produce an XML document from an SQL query. Also, this section demonstrated the idea that **XMLTABLE** is not limited to simple extraction and can be used for more complex queries of the XML data.

Hopefully, you are convinced that there is a great deal of value in the new SQL/XML support in DB2 for i 7.1. It makes it much easier to integrate XML and relational data together into an application. This section is only the beginning. The SQL/XML support in DB2 for i is substantial enough to receive its own book in the IBM i information center, and it can take some time and effort to master.

For more information about XML and RPG integration, see the following resources:

- ▶ *Accessing Web Services Using IBM DB2 for i HTTP UDFs and UDTFs*, found at:

https://www.ibm.com/partnerworld/wps/servlet/ContentHandler/stg_ast_sys_wp_access_web_service_db2_i_udf

- ▶ Replacing DB2 XML Extender with Integrated IBM DB2 for i XML Capabilities, found at:
[https://www.ibm.com/partnerworld/wps/servlet/ContentHandler?contentId=K\\$63TzTFkZwiPCA\\$cnt&roadMapId=Ib0toNReUYN4MDADrdm&roadMapName=Education+resources+for+IBM+i+systems&locale=en_US](https://www.ibm.com/partnerworld/wps/servlet/ContentHandler?contentId=K$63TzTFkZwiPCA$cnt&roadMapId=Ib0toNReUYN4MDADrdm&roadMapName=Education+resources+for+IBM+i+systems&locale=en_US)
- ▶ *SQL XML Programming*, found at:
<http://pic.dhe.ibm.com/infocenter/series/v7r1m0/topic/rzasp/rzasppdf.pdf>
- ▶ XML (i Can - Technical Tips for i):
http://ibmsystemsmag.blogs.com/i_can/xml/



Database re-engineering

Re-engineering the database is a vital component for the modernization process. But, you often cannot afford to throw away what you have today. This chapter is focused on the steps to help you re-engineer your database in place. Use what you have been using all along while using new technologies and methodologies. This chapter provides a step-by-step process that shows you the process and the tools to help you successfully navigate this important area.

9.1 Re-engineering versus re-representing

IBM has developed a process for refactoring databases that are created and modified over several decades to a new database that is created by using industry-standard Structured Query Language (SQL) Data Definition Language (DDL). The IBM database modernization strategy, through a series of phases, simplifies the refactoring of the current database to a new database. It is meant to be a highly iterative approach, allowing phases to overlap. The new database is defined by using SQL and contains the fundamental items that are required to establish data integrity and easily adapt to future business requirements. This phased approach is shown in Figure 9-1.

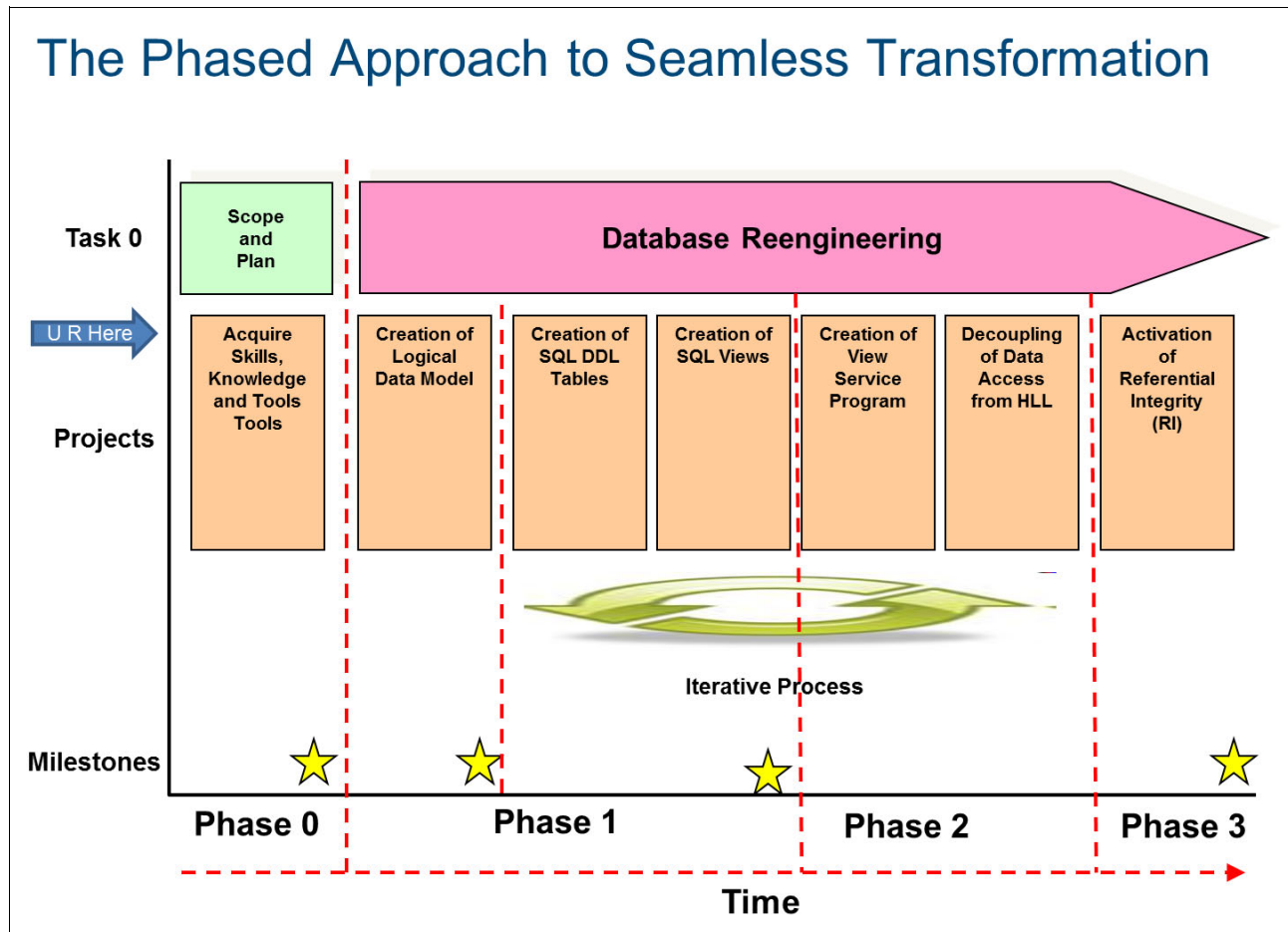


Figure 9-1 Overview of the database modernization phased approach

This section about database engineering explores each of these phases in more detail.

9.2 Getting started

This section provides basic considerations for re-engineering a database.

9.2.1 Business requirements that might result in re-engineering the database

The reasons for re-engineering a database differ for every company. However, here are some common reasons why you might consider re-engineering your database:

► Expansion into new markets

Expanding into other markets can be a great thing as a company grows. However, different countries pose various challenges from currency to address lines to language (CCSID). Older systems are not meant to handle these challenges and can cause a company to need to re-engineer their database to account for them.

► Advanced query reporting

Companies worldwide invest millions of dollars in operational applications to improve the way they conduct business. Although these systems provide significant benefits, clients often struggle to meet the needs of various users with analyzing the data that is collected and stored in those systems.

Back in the 1980s, IBM provided a utility to view data that is stored in S/36, S/38, and AS/400 databases. Now commonly referred to as Query/400, this product continues to this day to be a heavily used utility for viewing data, creating printed reports, or extracting data for purposes of moving the data into a more modern visualization or analysis tool (such as spreadsheets). Many of the previously mentioned operational applications still embed this decades old technology into their applications for reporting purposes. It is truly amazing that after more than 25 years that this utility is still so heavily used today by IT shops that are running IBM i on Power Systems to access data in DB2 for i.

Meanwhile, over those same decades, DB2 technology for processing data analysis requests has evolved *far* beyond what was available then. And, unfortunately, Query/400 has not kept up with the numerous inventions within DB2 for i to improve performance and management of the query environment. So, although it is still heavily used, it is not just the archaic front-end reporting writing interface that has fallen behind, but also the back-end processing by DB2 of queries that are submitted through the Query/400 tool.

In today's highly competitive economic environment, the company that can analyze data in operational system databases in a more fluid and understandable way can make more intelligent and timely decisions. The net result is business intelligence, which applies analytics to the data to provide insight for better business decisions.

Improved data analysis can provide many benefits, including the following items:

- Ability for users to get the data when they need it, and in the forms in which they need it, through a self-service model
- Allowing executives to monitor the state of the business through key performance indicator (KPI) dashboards
- Spot trends and exceptions in the data with real-time analytical processing
- Automation of the creation and delivery of static reports in PDF, spreadsheet, or formats that allow data to be further analyzed in smart mobile devices with 24x7 access

► 24x7 access

In a global economy, the window of opportunity for introducing change to meet business requirements has greatly decreased. In fact, for some customers, there is no window at all.

In addition, some customers must consolidate several systems in different locations to a single location to provide enterprise reporting.

Although IBM PowerHA® and an Operational Data Store (ODS) can meet HA and reporting requirements, as shown Figure 9-2, many businesses are hampered by outdated databases that do not contain appropriate attributes for consolidation.

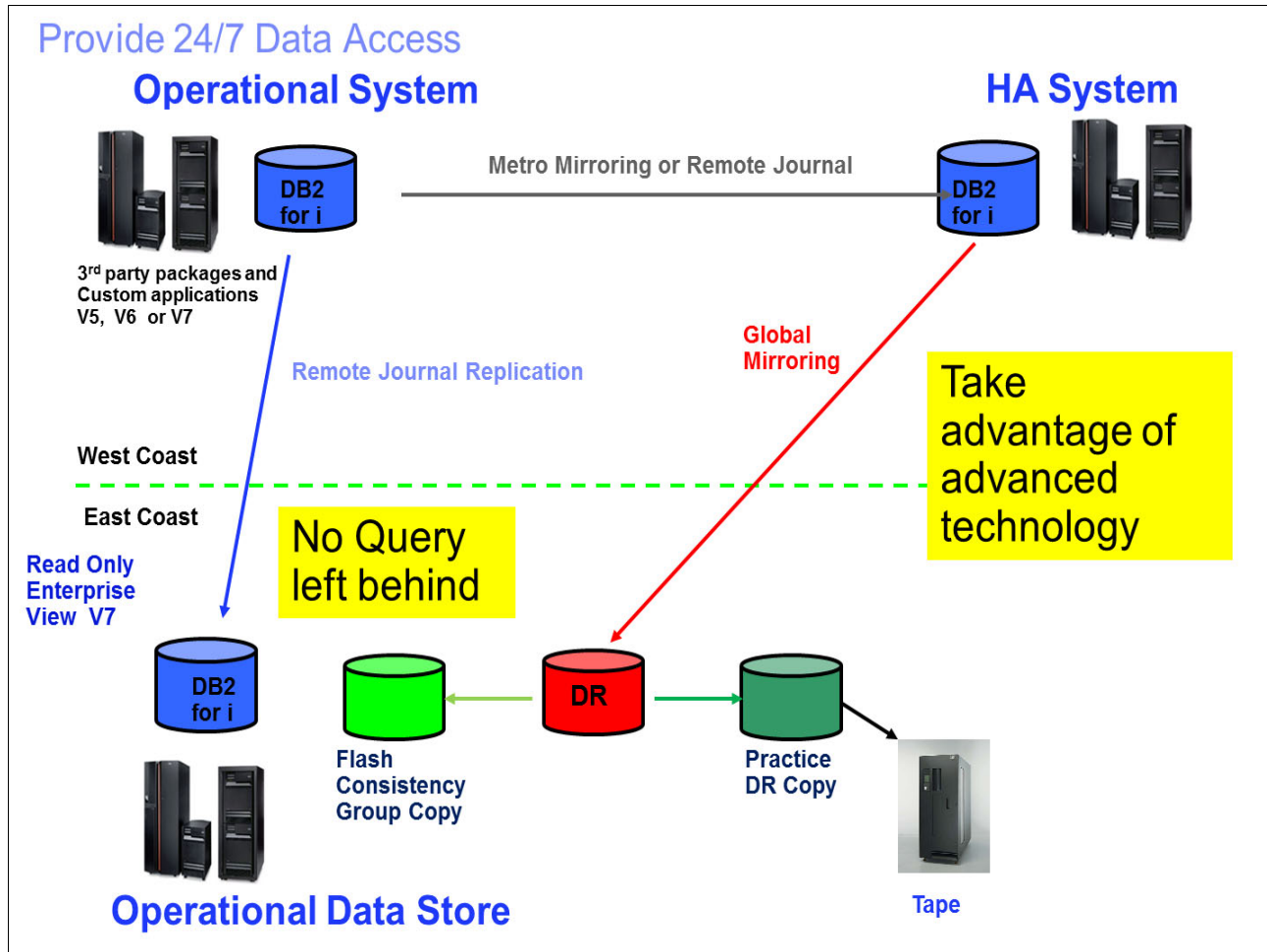


Figure 9-2 24x7 data access

Here are some of the most common characteristics of missing or less than optimal attributes:

- Double-byte Character Sets (DBCS)
- Multiple languages (CCSID)
- Lack of a surrogate primary key
- Lack of referential integrity
- ▶ Overcoming limits to growth

Many IBM i customers have been part of the IBM Rochester systems family for decades. These customers have reached, or are pushing the limits of, the database in terms of number of rows or maximum size of a table. The current limits are 4.2 billion rows or 1.7 TB per single table partition. In addition, it is not unusual for heritage databases to have issues storing larger numbers. Suddenly, you might need to increase the size of numeric columns because your sales numbers have expanded beyond the size of fields.

Re-engineering your database gives you the ability to go beyond the current table limitations and increase the size of a column by using an `ALTER TABLE ALTER COLUMN` statement. If you follow the outline of converting DDS to DDL that is laid out in this publication (for example, the usage of surrogate files), you do not have to recompile or change every program that accesses the file that the field being increased is in. You address only those programs that use the field that is being increased. All other programs that access the file but do not care about the field can be left untouched.

- Duplication of data, integrity rules, and business logic across applications

Over time, as more applications are added to the business portfolio, more attention has been given to application solutions, which, in many cases, involves putting data into applications that are not designed for it. Most solutions to this problem involve making copies of data.

Application development starts at the bottom, that is, the database itself. Without a solid foundation, the system becomes fragile, hard to manage, and eventually collapses under the weight of continued growth and application fixes. Solidifying the foundation is the first step to achieving agility, flexibility, and scalability.

Figure 9-3 shows the instability of an application-centric designed solution versus a data-centric designed solution.

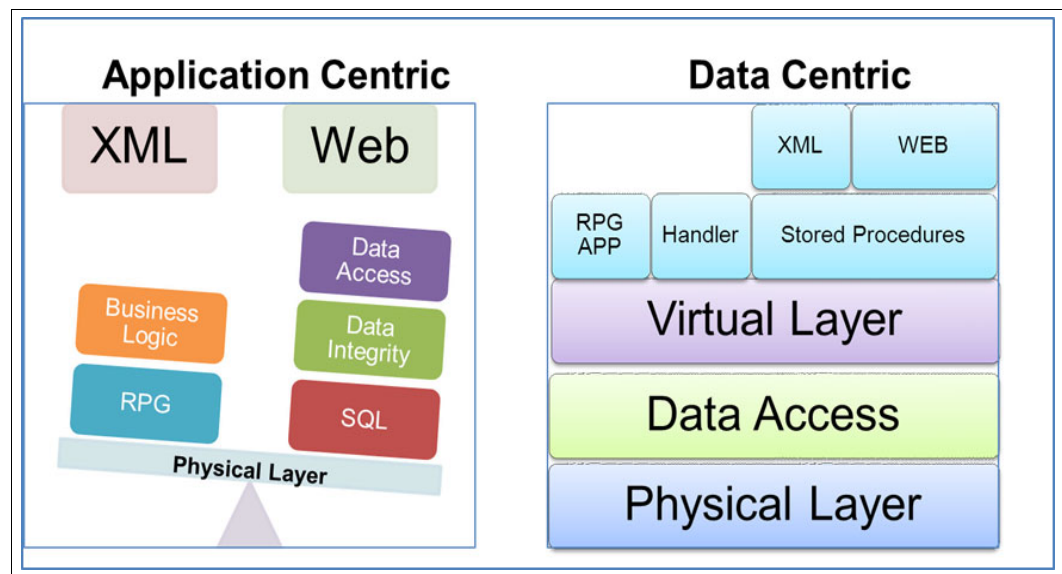


Figure 9-3 Top-down versus bottom-up database development

There are several key indicators that characterize application-centric environments:

- Loss of control

Application teams do things without informing IT, such as building applications using nonstandard storage engines that are not compatible with the enterprise database.

- Duplication of sensitive data across different databases

This occurs when enterprise data is copied directly from the production database and not “scrubbed” to eliminate sensitive data items (that is, credit card numbers, phone numbers, addresses, and so on) This data ends up in spreadsheets, eventually resulting in a loss of a single system of “truth”.

- Lack of security

All developers have access to the actual production data and can make copies on demand.

- Lack of data integrity

The database contains minimal constraints. Transactions that add, update, or delete the contents of the database are not running under commitment control. Valuable labor time is wasted restoring data from a backup when a transaction breaks in-flight.

- Intermittent system slowdowns

This occurs when applications become too top-heavy and more system resources are seized by the operating system to maintain an excessive number of supporting artifacts (that is, unnecessary indexes, wasted memory because of work files, costly table scans, and so on).

Data-centric programming provides a solution to the duplication of data issue by providing a central repository for the business rules, that is, the database. Data-centric development is characterized by the following conventions:

- Database capabilities are used to solve business problems. The database management system does most of the heavy lifting, using the strengths of the system
 - Ensure integrity of the data in the database by using DDL constraints (unique, no duplication, referential integrity between dependent relationships, and validation of column-level values)
 - Implementation of shared business logic in the database by using SQL triggers and functions
 - Separation of the business logic from the high-level application programs by using stored procedures that are written in SQL or host-based languages (that is, RPG) containing SQL statements (also known as external stored procedures)
- Eliminating or improving batch applications

One of the first indicators of possible database issues are batch applications that are no longer completing in their designated window of time. Many of these applications have not changed in decades and are using traditional record-at-a-time data access methods to process tens or hundreds of millions of rows. These jobs typically run overnight or on weekends and have an impact on daytime productivity. In addition, they are a deterrent to providing 24x7 operation support.

Some of these batch jobs were developed 30 years ago when interactive response time was critical. Today, the need for subsecond response time is secondary to the need for throughput. It is not uncommon to find that many IBM i systems are running at 100% usage during the nighttime, and the system resources are going unused during daytime hours.

In addition, many batch applications make temporary copies of data, another carryover from traditional programming. Today, many advanced techniques are available by using SQL and data-centric programming to eliminate data duplication in batch.

9.2.2 Training, teaming, and tools

This section describes training, teaming, and tools.

A database engineer or a database administrator

DB2 for i provides an enormous amount of database administration support. The DBMS creates temporary indexes that it needs to support queries that are running on the system, replan queries currently running that it believes take too long, and automatically rebuild indexes as needed. With DB2 for i running on Power Systems, you have a database administrator (DBA) available to you full-time to help with the system.

However, you might still need a part-time or full-time database engineer (DBE). The DBE also plays one or more of the following roles, depending on how large your IBM i shop is:

- ▶ Works with the application staff

A DBE creates naming standards for tables, indexes, views, columns, and so on. The DBE creates standards for usage of tables and indexes. For example, they might determine that new SQL objects should be used in File Definition Specs (F Specs) or that **SELECT * FROM . . .** in SQL statements should not be used. The DBE works with application staff to teach SQL, enforce SQL rules, and help with performance tuning.

- ▶ Architect

The DBE uses design tools such as IBM InfoSphere Data Architect or IBM Data Studio to lay out the relationships between entities (both physical and logical).

- ▶ Analysis

The DBE evaluates the temporary indexes that the system has created and determines whether these indexes should be made permanent. In addition, the DBE diagnoses performance issues and evaluates SQL DBMS issues.

Suggested framework

Here is a more detailed description of the suggested framework for re-engineering:

- ▶ Physical data layer

The physical data layer consists of the physical tables, constraints, and indexes. With few exceptions, the physical layer should not be referenced directly in programs. The physical layer is designed to hold the data and ensure the integrity of the relationships between tables by using constraints.

The physical layer should also be normalized to at least third normal form, and have identity columns as the primary key. Unique requirements on business keys should be by using unique index constraints.

- ▶ Virtual or data abstract layer

The virtual layer consists of the views that bring various pieces of data together across physical tables. Views can be used to de-normalize the data into a format that the application needs. For example, you might have an ORDER_HEADER and ORDER_DETAIL table at the physical data layer and a view that brings specific fields from ORDER_HEADER together with specific fields with ORDER_DETAIL so a program can display these fields to the user.

Views are also used to provide aggregate information that should *not* be stored at the physical layer. For example, a total order amount for a specific order should *not* be stored in the ORDER_HEADER table. A view can sum the ORDER_DETAIL lines to give a count of lines and total amount of the order.

This is what a database engineer and a database architect can help design so that the programs do not have to track information that the DBMS can do for you.

- ▶ Data access layer

The data access layer consists of the stored procedures and service programs that provide a central point of writing, updating, and deleting of the data. In general, there should be no reason for a program to write, update, or delete data within the program without going through a data access application. Programs read the data through the logical data layer. However, it is not a bad idea to have a data access layer for reading the data also. This configuration allows a stored procedure to return a result set to multiple different applications. For more information about reading result sets from stored procedures, see Chapter 8, “Data-centric development” on page 221.

This provides a central touch-point to the DBMS that can be managed.

- ▶ Application layer

At the application layer are the programs that process and display the information to the user community. This can be standard green-screen programs, but it also can include PHP, .NET, and Java applications. These programs read and write information through the data access layer, providing a standardized answer that is given to all applications regardless of the language or user interface (UI) involved.

9.3 The phased approach to seamless transformation

The Phase 0 discovery process consists of a database, skills, and tools inventory assessment. This assessment must be done to identify current strengths and weaknesses within the current database structures and the development staff.

The assessment should answer the following questions:

- ▶ How much do you know?
- ▶ What tools do you have?
- ▶ Do you use them?
- ▶ What do you need?
 - Education
 - Tools
 - More resources

Phase 1 can be accomplished with little or no change to existing programs. Phase 2 isolates the database, allowing future changes to the physical database without impacting new, SQL developed data access modules. Phase 3 incorporates data integrity, permissions, and encryption rules into the database, and protects the database from harm. The phases are described in more detail in the following sections.

9.3.1 Phase 0: Discovery

The current database might not have been designed with a relational database in mind. Many relationships might exist that are not explicitly declared through Referential Integrity (RI) constraints. The database modernization strategy should focus on these relationships, implicit and explicit, and identify where they exist so that the correct RI data integrity rules can be applied.

Figure 9-4 on page 359 shows the current state of the host-based application and database access.

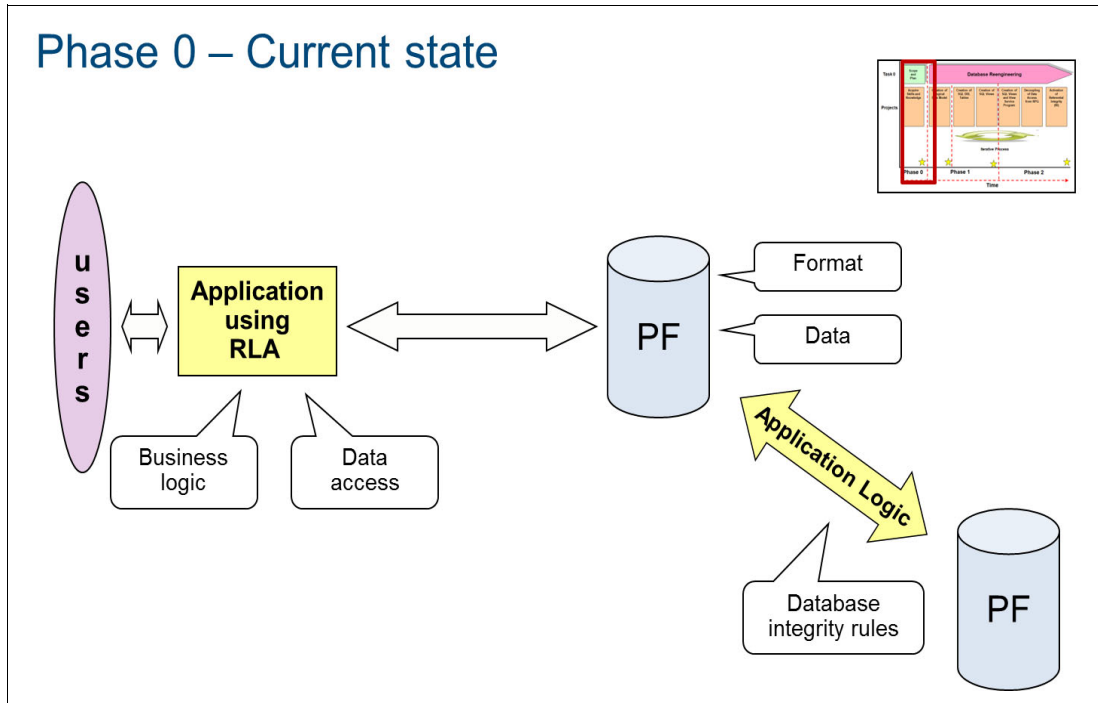


Figure 9-4 Phase 0 - understanding the current state

Each file must be reviewed to ensure that it contains the core fundamental items that you expect to find in a well-designed relational database.

The following core items must be present in the current physical data model:

- ▶ Core requirements
 - Each table is defined by using SQL Data Definition Language (DDL).
 - The table contains a unique key constraint (also known as the public key).
 - If the table is the parent in a parent-child relationship, it contains a primary key constraint (also known as the surrogate key).
 - The table contains time stamp columns that are used to identify the data and time the row was inserted into the table and when the last change occurred to the row.
 - The table is being logged by using journal support.
- ▶ Secondary requirements
 - The table contains a single partition (member) if the following conditions are true:
 - The total number of rows are less than 4.2 billion.
 - The total size of the partition is less than 1.7 TB.
 - The table is not cleared, reorganized, or copied into.

Physical files that do not meet the preceding criteria are considered to be “out of compliance” with core standards.

It is important to understand the skills of individuals who are doing the work of modernization in addition to the tools that are available to use. For those lacking specific skills, IBM provides various classes to help fill the skills gap. Here is a link to the DB2 for i education offerings:

<http://www.ibm.com/systems/power/software/i/db2/education/index.html>

The following tools are available to help with the modernization process:

- ▶ IBM i Navigator
- ▶ IBM Rational Development Suite, which includes the following components:
 - Rational Developer for IBM i (Rational Developer for i V9.0)
 - IBM Data Studio 4 (A no charge download), found at:
 - <http://www.ibm.com/developerworks/downloads/im/data/>
- ▶ Third-party automation tools: Xcase, X-Analysis, AO Foundation, and so on.

Figure 9-5 is the physical data model that represents the current state of the Flight/400 database. We created this model with the IBM Data Studio. For more information about IBM Data Studio, see 8.6.1, “Managing DB2 for i with IBM Data Studio” on page 291.



Figure 9-5 Example Flight/400 database - physical data model

In the diagram, apparently there are no unique or primary key constraints or any declared relationships (that is, referential integrity). We used the DB2 for i system catalog views to validate these assumptions.

System catalog views

System catalog views (SCV) are SQL views that are in QSYS2 and are designed over the physical catalog tables that are in QSYS. The views contain metadata about the database objects in various libraries.

Note: Metadata is information about information. It serves to describe data, and its usage is not limited to the field of SQL or information technology. Most, if not all, SQL-based RDBMS allow the extraction of the metadata of their content. For example, metadata is important to reverse engineer databases. Database design tools typically use metadata to display database models.

Some of the SCVs are built over the system tables in QSYS (sometimes called Database Cross Reference Files). Other SCVs are UDTFs that access information through system APIs. Table 9-1 shows the database cross-reference files that the catalog views are attached over.

Table 9-1 Example system catalog view

QADBCST	QADBKFLD	QADBXSFLD
QADBFDEP	QADBPKG	QADBXRIGB
QADBF CST	QADBXRDBD	QADBXRIGC
QADBIFLD	QADB XREF	QADBXRIGD

Note: These tables are not shipped with the SELECT privilege to PUBLIC and should not be used directly.

Using the SCV, you can find all the information that you need about PF, LF, SQL Tables, Indexes, and Views and Stored Procedures and Triggers. The SCV allows you to query all the column definitions or table definitions in a specific library. The SCV allows you to see the relationships between dependent objects.

The SCV can play a pivotal role in reverse engineering your DDS objects and converting them to SQL DDL.

Table 9-2 gives the specific catalog view name and its definition.

Table 9-2 Catalog view names and definitions

Catalog view	Description
SYSCATALOGS	The SYSCATALOGS view contains one row for each relational database to which a user can connect.
SYSCHKCST	The SYSCHKCST view contains one row for each check constraint in the SQL schema.
SYSCOLUMNS	The SYSCOLUMNS view contains one row for every column of each table and view in the SQL schema (including the columns of the SQL catalog).
SYSCOLUMNS2	The SYSCOLUMNS2 view contains one row for every column of each table and view in the SQL schema (including the columns of the SQL catalog).
SYSCOLUMNSTAT	The SYSCOLUMNSTAT view contains one row for every column in a table partition or table member that has a column statistics collection. If the table is a distributed table, the partitions that are on other database nodes are not contained in this catalog view.
SYSCST	The SYSCST view contains one row for each constraint in the SQL schema.
SYSCSTCOL	The SYSCSTCOL view records the columns on which constraints are defined. There is one row for every column in a unique, primary key, and check constraint and the referencing columns of a referential constraint.
SYSCSTDEP	The SYSCSTDEP view records the tables on which constraints are defined.
SYSFIELDS	The SYSFIELDS view contains one row for every column that has a field procedure.
SYSFUNCS	The SYSFUNCS view contains one row for each function that is created by the CREATE FUNCTION statement.
SYSINDEXES	The SYSINDEXES view contains one row for every index in the SQL schema that is created by using the SQL CREATE INDEX statement, including indexes on the SQL catalog.
SYSINDEXSTAT	The SYSINDEXSTAT view contains one row for every SQL index.

Catalog view	Description
SYSJARCONTENTS	The SYSJARCONTENTS table contains one row for each class that is defined by a jarid in the SQL schema.
SYSJAROBJECTS	The SYSJAROBJECTS table contains one row for each jarid in the SQL schema.
SYSKEYCST	The SYSKEYCST view contains one or more rows for each UNIQUE KEY, PRIMARY KEY, or FOREIGN KEY in the SQL schema. There is one row for each column in every unique or primary key constraint and the referencing columns of a referential constraint.
SYSKEYS	The SYSKEYS view contains one row for every column of an index in the SQL schema, including the keys for the indexes on the SQL catalog.
SYSMQTSTAT	The SYSMQTSTAT view contains one row for every materialized table partition.
SYSPACKAGE	The SYSPACKAGE view contains one row for each SQL package in the SQL schema.
SYSPACKAGESTAT	The SYSPACKAGESTAT view contains one row for each SQL package in the SQL schema.
SYSPARMS	The SYSPARMS table contains one row for each parameter of a procedure that is created by the CREATE PROCEDURE statement or function that is created by the CREATE FUNCTION statement. The result of a scalar function and the result columns of a table function are also returned.
SYSPARTITIONDISK	The SYSPARTITIONDISK view contains one row for every disk unit that is used to store data of every table partition or table member. If the table is a distributed table, the partitions that are on other database nodes are not contained in this catalog view. They are contained in the catalog views of the other database nodes.
SYSPARTITIONINDEXDISK	The SYSPARTITIONINDEXDISK view contains one row for every disk unit that is used to store the index data of every table partition or table member. If the index is a distributed index, the partitions that are on other database nodes are not contained in this catalog view. They are contained in the catalog views of the other database nodes.
SYSPARTITIONINDEXES	The SYSPARTITIONINDEXES view contains one row for every index that is built over a table partition or table member. If the table is a distributed table, the indexes over partitions that are on other database nodes are not contained in this catalog view. They are contained in the catalog views of the other database nodes.

Catalog view	Description
SYSPARTITIONINDEXSTAT	The SYSPARTITIONINDEXSTAT view contains one row for every index that is built over a table partition or table member. Indexes that share another index's binary tree are not included. If the table is a distributed table, the indexes over partitions that are on other database nodes are not contained in this catalog view. They are contained in the catalog views of the other database nodes.
SYSPARTITIONMQTS	The SYSPARTITIONMQTS view contains one row for every materialized table that is built over a table partition or table member. If the table is a distributed table, the materialized tables over partitions that are on other database nodes are not contained in this catalog view. They are contained in the catalog views of the other database nodes.
SYSPARTITIONSTAT	The SYSPARTITIONSTAT view contains one row for every table partition or table member. If the table is a distributed table, the partitions that are on other database nodes are not contained in this catalog view. They are contained in the catalog views of the other database nodes.
SYSPROCS	The SYSPROCS view contains one row for each procedure that is created by the CREATE PROCEDURE statement.
SYSPROGRAMSTAT	The SYSPROGRAMSTAT view contains one row for each program, service program, and module that contains SQL statements.
SYSREFCST	The SYSREFCST view contains one row for each foreign key in the SQL schema.
SYSROUTINEDEP	The SYSROUTINEDEP view records the dependencies of routines.
SYSROUTINES	The SYSROUTINES table contains one row for each procedure that is created by the CREATE PROCEDURE statement and each function that is created by the CREATE FUNCTION statement.
SYSSCHEMAS	The SYSSCHEMAS view contains one row for every schema in the relational database.
SYSSEQUENCES	The SYSSEQUENCES view contains one row for every sequence object in the SQL schema.
SYSTABLEDEP	The SYSTABLEDEP view records the dependencies of materialized query tables.
SYSTABLEINDEXSTAT	The SYSTABLEINDEXSTAT view contains one row for every index that has at least one partition or member that is built over a table. If the index is over more than one partition or member, the statistics include all those partitions and members. If the table is a distributed table, the partitions that are on other database nodes are not included. They are contained in the catalog views of the other database nodes.

Catalog view	Description
SYSTABLES	The SYSTABLES view contains one row for every table, view, or alias in the SQL schema, including the tables and views of the SQL catalog.
SYSTABLESTAT	The SYSTABLESTAT view contains one row for every table that has at least one partition or member. If the table has more than one partition or member, the statistics include all partitions and members. If the table is a distributed table, the partitions that are on other database nodes are not included. They are contained in the catalog views of the other database nodes.
SYSTRIGCOL	The SYSTRIGCOL view contains one row for each column that is either implicitly or explicitly referenced in the WHEN clause or the triggered SQL statements of a trigger.
SYSTRIGDEP	The SYSTRIGDEP view contains one row for each object that is referenced in the WHEN clause or the triggered SQL statements of a trigger.
SYSTRIGGERS	The SYSTRIGGERS view contains one row for each trigger in an SQL schema.
SYSTRIGUPD	The SYSTRIGUPD view contains one row for each column that is identified in the UPDATE column list, if any.
SYSTYPES	The SYSTYPES table contains one row for each built-in data type and each distinct type and array type that is created by the CREATE TYPE statement.
SYSVARIABLEDEP	The SYSVARIABLEDEP table records the dependencies of variables.
SYSVARIABLES	The SYSVARIABLES table contains one row for each global variable.
SYSVIEWDEP	The SYSVIEWDEP view records the dependencies of views on tables, including the views of the SQL catalog.
SYSVIEWS	The SYSVIEWS view contains one row for each view in the SQL schema, including the views of the SQL catalog.
XSRANNOTATIONINFO	The XSRANNOTATIONINFO table contains one row for each annotation in an XML schema to record the table and column information about the annotation.
XSROBJECTCOMPONENTS	The XSROBJECTCOMPONENTS table contains one row for each component (document) in an XML schema.
XSROBJECTHIERARCHIES	The XSROBJECTHIERARCHIES table contains one row for each component (document) in an XML schema to record the XML schema document hierarchy relationship.
XSROBJECTS	The XSROBJECTS table contains one row for each registered XML schema.

For more information about system catalog views, see the IBM i 7.1 information center at the following website:

<http://pic.dhe.ibm.com/infocenter/iseriess/v7r1m0/topic/db2/rbafzcatalog.htm>

Examples of using system catalog views

The following code examples show various uses for the system catalog view.

To get a list of physical files and SQL tables in a library, use the code that is shown in Example 9-1.

Example 9-1 Getting a list of physical files and SQL tables in a library

```
SELECT *
  FROM QSYS2.SYSTABLES
  WHERE TABLE_SCHEMA = 'mylibrary' AND TABLE_TYPE
  IN ("P", "T");
```

To get a list of all columns for a specific physical file, use the code that is shown in Example 9-2.

Example 9-2 Getting a list of all columns for a specific Physical File

```
SELECT *
  FROM QSYS2.SYSCOLUMNS
  WHERE TABLE_SCHEMA = 'mylibrary' AND TABLE_NAME
  = 'mytable'
  ORDER BY ORDINAL_POSITION;
```

To get a list of tables in a specific library that deleted rows and might need to be re-engineered, use the code that is shown in Example 9-3.

Example 9-3 Getting a list of tables in a specific library that deleted rows

```
SELECT TABLE_SCHEMA, TABLE_NAME, NUMBER_ROWS, NUMBER_DELETED_ROWS
  FROM QSYS2.SYSTABLESTAT
  WHERE TABLE_SCHEMA = 'mylibrary';
```

Note: Each of these SQL queries can also be put into a view and made permanent on disk. This process allows you to automate specific tasks that keep your system clean. For example, you can enhance the query in Example 9-3 to calculate the percentage of deleted rows and set up a process to automatically reorganize the tables that meet specific delete percentage criteria.

Example 9-4 shows how to get a summary list of indexes by type within a schema to determine core fundamental compliance.

Example 9-4 SQL statement to get the summary list of indexes by type within a schema

```
WITH Index_Counts AS(
  SELECT T1.*, DECIMAL(Sparse,1,0) Sparse_Indexes,DECIMAL(derived_key,1,0)
  Derived_Indexes,
  CASE SMALLINT(partitioned) WHEN 0 THEN 1 ELSE 0 END Part_Indexes,
  CASE SMALLINT(partitioned) WHEN 1 THEN 1 ELSE 0 END Not_Partitioned,
  CASE SMALLINT(partitioned) WHEN 2 THEN 1 ELSE 0 END Part_LF,
  CASE SMALLINT(ACCPH_TYPE) WHEN 0 THEN 1 ELSE 0 END Max_1TB,
  CASE SMALLINT(ACCPH_TYPE) WHEN 1 THEN 1 ELSE 0 END Max_4GB,
```

```

CASE SMALLINT(ACCPH_TYPE) WHEN 2 THEN 1 ELSE 0 END EVI,
CASE SMALLINT(UNIQUE) WHEN 0 THEN 1 ELSE 0 END Unique_IDX,
CASE SMALLINT(UNIQUE) WHEN 1 THEN 1 ELSE 0 END Unique_NOT_NULL,
CASE SMALLINT(UNIQUE) WHEN 2 THEN 1 ELSE 0 END FIFO_IDX,
CASE SMALLINT(UNIQUE) WHEN 3 THEN 1 ELSE 0 END LIFO_IDX,
CASE SMALLINT(UNIQUE) WHEN 4 THEN 1 ELSE 0 END FCFO_IDX
from QSYS2.SYSTABLEINDEXSTAT T1)
select IFNULL(Table_Schema,'Final') File_Lib ,
       IFNULL(Index_Schema,'Schema') Index_Lib ,
       IFNULL(Index_Type, 'Totals') Index_Type,
       COUNT(*) Idx_Type_Count,
       COUNT(DISTINCT Table_Name) Tables,
       (COUNT(*)/COUNT(DISTINCT Table_Name))
       Avg_Idx_Per_Tbl,
       MAX(Number_Key_Columns) Max_Keys,
       MAX>Last_Query_Use) Last_Qry_Use,
       MAX>Last_Statistics_Use) Last_Stats_Use,
       SUM(Query_Use_Count) Qry_Use,
       SUM(Query_Statistics_Count) Stats_Use,
       MAX(last_used_timestamp) Last_Use,
       MAX(days_used_count) Max_Days_Used,
       SUM(Logical_Reads) Lgl_Reads,
       SUM(Physical_Reads) Phy_Reads,
       SUM(Sequential_Reads) Seq_Reads,
       SUM(Random_Reads) Random_Reads,
       IFNULL(MIN(Logical_Page_Size),4096)
       Min_Page_Size,
       IFNULL(MAX(Logical_Page_Size),32768)
       Max_Page_Size,
       MAX(index_size) Max_Idx_Size,
       SUM(Sparse_Indexes) Sparse_Indexes,
       SUM(Derived_Indexes) Derived_Indexes,
       SUM(Part_Indexes) Part_Indexes,
       SUM(Not_Partitioned) NP_Indexes,
       SUM(Part_LF) Part_LF,
       SUM(Max_1TB) Max_1TB,
       SUM(Max_4GB) Max_4GB,
       SUM(EVI) EVI, SUM(Unique_IDX) Unique_IDX,
       SUM(Unique_NOT_NULL) Unique_NOT_NULL,
       SUM(FIFO_IDX) FIFO_IDX,
       SUM(LIFO_IDX) LIFO_IDX,
       SUM(FCFO_IDX) FCFO_IDX
from Index_Counts T1
WHERE table_schema = 'FLGHT400'
GROUP BY ROLLUP (Table_Schema, Index_Schema, INDEX_TYPE);

```

Figure 9-6 on page 367 shows a snippet from the result set of the query in Example 9-4 on page 365.

FILE_LIB	INDEX_LIB	INDEX_TYPE	IDX_TYPE_COUNT	TABLES	AVG_IDX_PER_TBL	MAX_KEYS
FLGHT400	FLGHT400	LOGICAL	15	7	2	4
FLGHT400	FLGHT400	PHYSICAL	13	13	1	1
FLGHT400	FLGHT400	Totals	28	13	2	4
FLGHT400	Schema	Totals	28	13	2	4
Final	Schema	Totals	28	13	2	4

Figure 9-6 Output from the SQL statement in Example 9-4 on page 365

Example 9-5 shows an SQL statement to get a summary list of table statistics by schema to determine file usage characteristics.

Example 9-5 SQL statement for getting a summary of table statistics

```

SELECT IFNULL(TS.Table_Schema,'Totals') Schema,
       COUNT (DISTINCT TS.Table_Name) total_files, MAX(NUMBER_ROWS) MAX_ROWS,
       MAX(number_deleted_rows) max_deletes, MAX(DECIMAL(DATA_SIZE*.000000001,9,5))
       MAX_SIZE_GB, MAX(Number_Partitions) MaxMembers,
       SUM(Open_Operations) Opens,
       SUM(Close_Operations) Closes,
       SUM(Insert_Operations) Writes,
       SUM(Update_Operations) Updates, SUM(Delete_Operations) Deletes,
       SUM(Clear_Operations) Clears,
       SUM(Copy_Operations) Copies, SUM(Reorganize_Operations) Reorgs,
       SUM(Index_Builds) IxBuilds,
       SUM(Logical_Reads) LglReads,
       SUM(Physical_Reads) PhyReads
FROM QSYS2.systablestat TS
JOIN QSYS2.systables ST
  ON TS.table_schema = ST.table_schema
  AND TS.table_name = ST.table_name
WHERE ST.FILE_TYPE <> 'S' --Omit Source Physical Files
GROUP BY ROLLUP (TS.Table_Schema);

```

Figure 9-7 shows a snippet from the result set of the query in Example 9-5.

SCHEMA	TOTAL_FILES	MAX_ROWS	MAX_DELETES	MAX_SIZE_GB	MAXMEMBERS	OPENS	CLOSES	WRITES
FLGHT400	17	701666	12	0.12793	1	0	0	0
Totals	17	701666	12	0.12793	1	0	0	0

Figure 9-7 Output from the SQL statement in Example 9-5

9.3.2 Phase 1: Transformation

The Phase 1 process begins by reviewing the current state physical data model (PDM), from Phase 0, to ensure that relationships are correctly defined and to verify that work files are identified. The PDM can then be used to complete the optional Logical Data Model (LDM). The LDM is your blueprint for all database enhancements going forward. It is a preferred practice that you use a graphical data modeling tool to create the LDM as opposed to using presentation or drawing tools.

The construction tasks of Phase 1 can begin before the creation of the LDM. The construction tasks are repetitive and ongoing. In some cases, these processes can be automated, in other cases they must be done painstakingly by hand. In either case, the strategy is meant to break a monumental undertaking into multiple, manageable projects. Each project can then be tackled one at a time, or in parallel by multiple teams. The Phase 1 database re-engineering milestone is complete when all tables are deployed

Figure 9-8 shows the main goals of Phase 1.

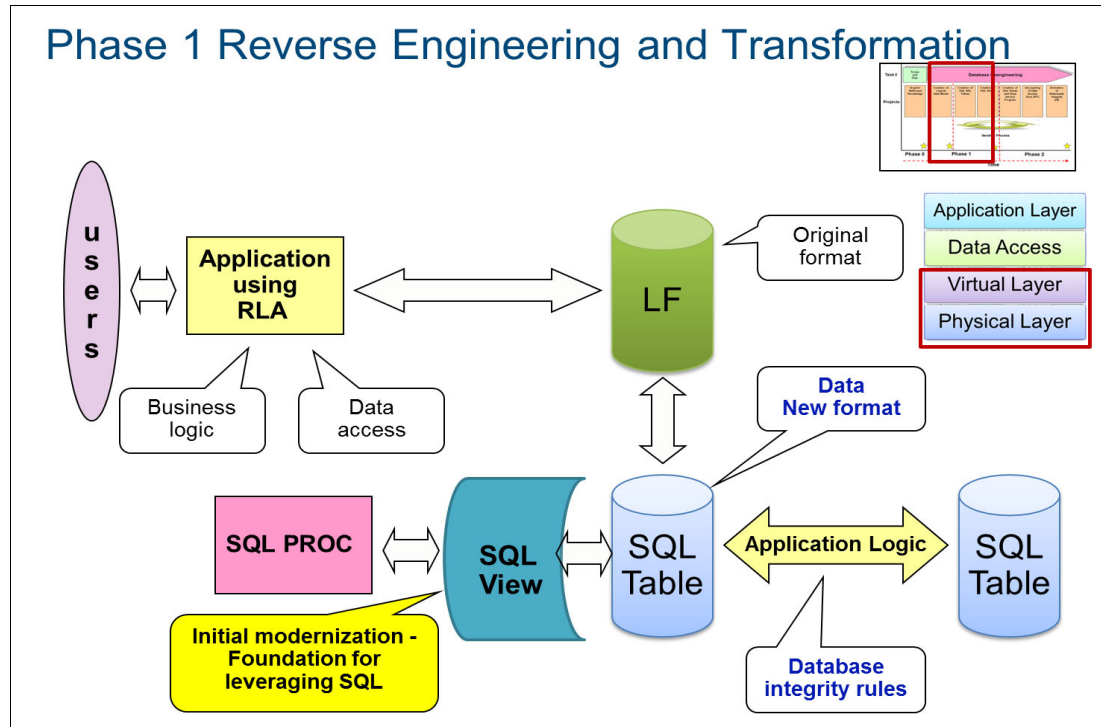


Figure 9-8 Phase 1 - Reverse Engineering and Transformation

Here is a list of tasks that are completed as part of Phase 1:

- ▶ Create an LDM (optional).
- ▶ Create the DDL for the new SQL tables.
- ▶ Create surrogate files, indexes, and views.

Creating and updating the logical data model

If you do not have a data modeling tool that provides logical data modeling capabilities, you can proceed to “Creating the DDL for the new SQL tables” on page 374.

The creation of the LDM begins with the initial pair of candidate files to be re-engineered. The future state LDM is updated as these files are corrected and more files are added to the LDM.

Figure 9-9 on page 369 shows the data modeling process that we used for Phase 1 for our example.

Creating an LDM from an Existing Physical Data Model

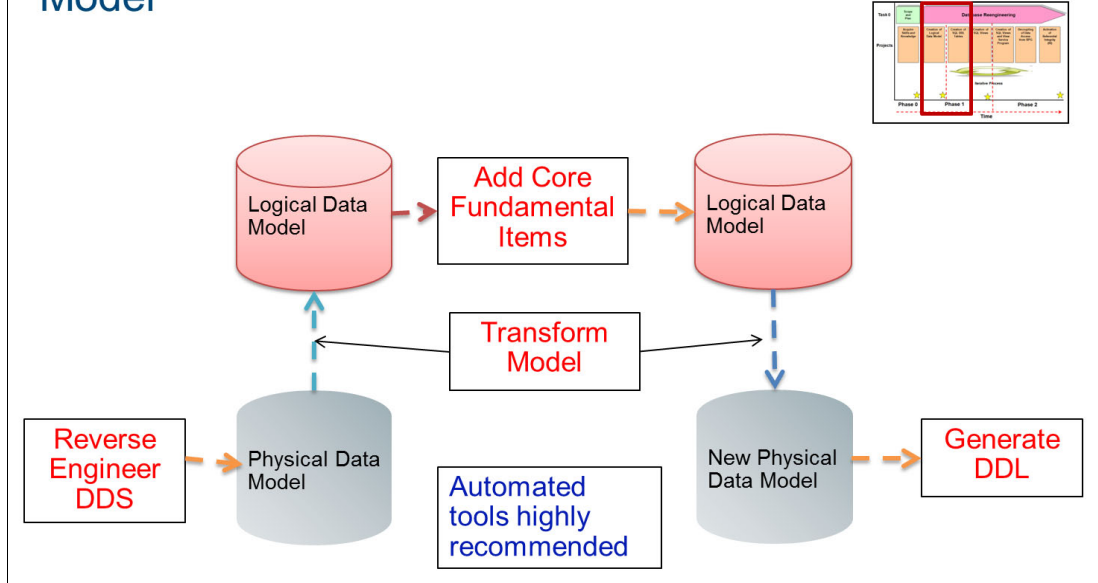


Figure 9-9 Creating an LDM from an existing physical data model

Figure 9-10 is an entity relationship diagram (ERD) that is generated from an LDM that represents the future state of our example Flights/400 database.

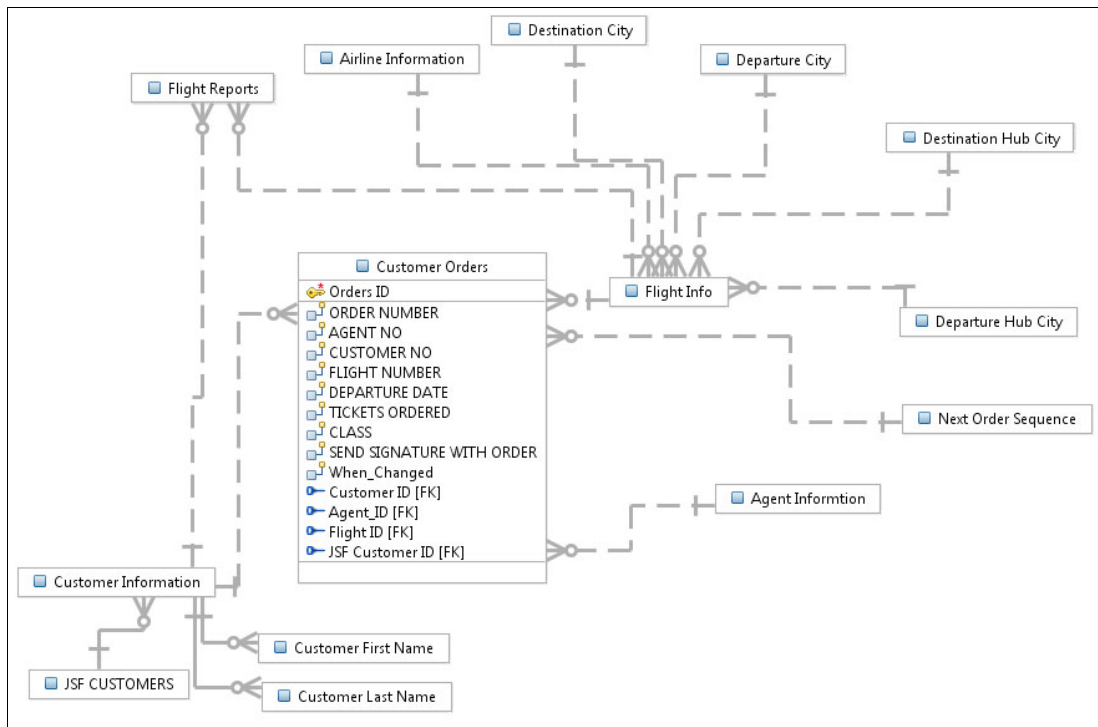


Figure 9-10 Flight/400 database entity relationship diagram

Here are the subtasks that are done during the creation of the LDM:

- ▶ Create and update the domain model with the core fundamental items.
- ▶ Create and update the glossary model with naming standards and abbreviations.
- ▶ Add the core fundamental items to the LDM entities.
- ▶ Transform the LDM to a physical data model.

Creating and updating the domain model with core fundamental items

One way to ensure compliance of field names, attributes, and so on, is to take advantage of domain support (also known as a data dictionary). A domain model contains the permissible attributes for commonly used columns. A good starting point is to create a domain model that contains the attributes of the primary key and row change time stamp columns.

Figure 9-11 contains an example of a domain model that is created with the InfoSphere Data Architect tool. Each data type is defined at its lowest level, or atomic level, in essence a column is not a composite of several types. For example, LONG_ID defines a primary key that has a base type of LONG (BIGINT in DB2 for i) as opposed to defining a column named ID and then choosing an attribute during the creation of the new physical data model.

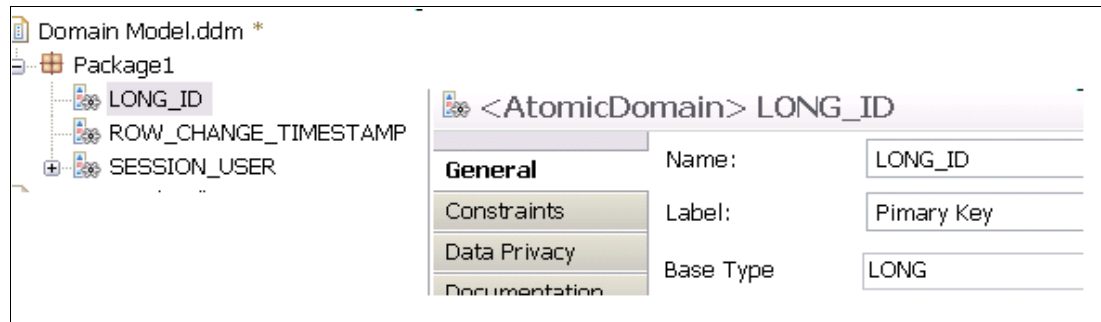


Figure 9-11 InfoSphere Data Architect domain model

Identity column domain

Every parent table in the LDM must have a primary key. This primary key is also the identity column for this table. The base type of the column is numeric and one of the following types:

- ▶ LONG (BIGINT): Use this type for transactional tables, archive tables, or those tables where the number of rows exceed 2,147,483,647.
- ▶ INTEGER: Use this type for master tables where the number of rows is greater than 32,767 and less than 2,147,483,648.
- ▶ SHORT (SMALLINT): Use this type for validation or master tables where the number of rows never exceed 32,767 rows.
- ▶ CHAR: This type is not recommended as an identity column because it cannot be automatically generated. It can be designated as a primary key in lookup or reference tables where the value is understood (for example, state codes).

In addition, identity columns should have the attributes primary key, required, and it is a surrogate key (automatically generated except for CHAR columns). Figure 9-12 on page 371 shows how this is specified for the column by using InfoSphere Data Architect.

<Attribute> HeadB_ID	
General	Data Type: 'LONG
Type	
Volumetrics	Length...cision:
Documentation	Scale:
Annotation	Default Value:
Advanced	<input checked="" type="checkbox"/> Primary Key <input checked="" type="checkbox"/> Required <input checked="" type="checkbox"/> Surrogate Key

Figure 9-12 InfoSphere Data Architect identity column attributes

Data change time stamp domain

Each table should contain a data change time stamp column, referred to as the Row Change Timestamp, to reflect the date and time of the last change.

The data change time stamp domain has a base type of `TIMESTAMP`. The column that is referencing the Data Change Timestamp Domain must have a default of `CURRENT_TIMESTAMP`.

Domain candidates

The current state database can contain several columns, which represent the same type of data. Possible examples can be address information (city, state, country, and so on), and name (customer, agent, and airline). A domain provides a method for the consistent usage of attributes for like pieces of data

For example, the `FLIGHT400` database contains both an agent name and customer name column. Both are defined as `VARCHAR(64)`. A domain that is named `PERSON_NAME` and defined as `VARCHAR(64) ALLOCATE(30)` and used to define both `CUSTOMER_NAME` and `AGENT_NAME` ensures that either column (or a new column such as `PASSENGER_NAME`) cannot inadvertently be defined with an inconsistent attribute.

Another benefit of using domains is that changing the domain attributes automatically changes the attributes of the columns that are referencing the domain.

Example 9-6 is an example of a query that is used to analyze the `Flight/400` columns to identify domain candidates:

Example 9-6 Example query to identify domain candidates for the `Flight/400` database

```

select column_name, data_type, length, count(*) column_count
from qsys2.syscolumns SC
join qsys2.systables ST USING(table_schema, table_name)
where table_schema = 'FLIGHT_DB2'
      and table_type IN ('P', 'T')
      and column_name LIKE '%NAME%'
group by column_name, data_type, length
ORDER BY length desc;

```

Creating a glossary model with naming standards

The InfoSphere Data Architect Glossary Model is another tool that is available to assist with consistent naming standards, abbreviations, compliance, and documentation. The glossary contains industry standards for abbreviations. Abbreviations that are used by Flight/400 can be added to the Glossary Model and then used for naming conventions.

Figure 9-13 contains a sample of the Enterprise Glossary Model that is included with InfoSphere Data Architect.

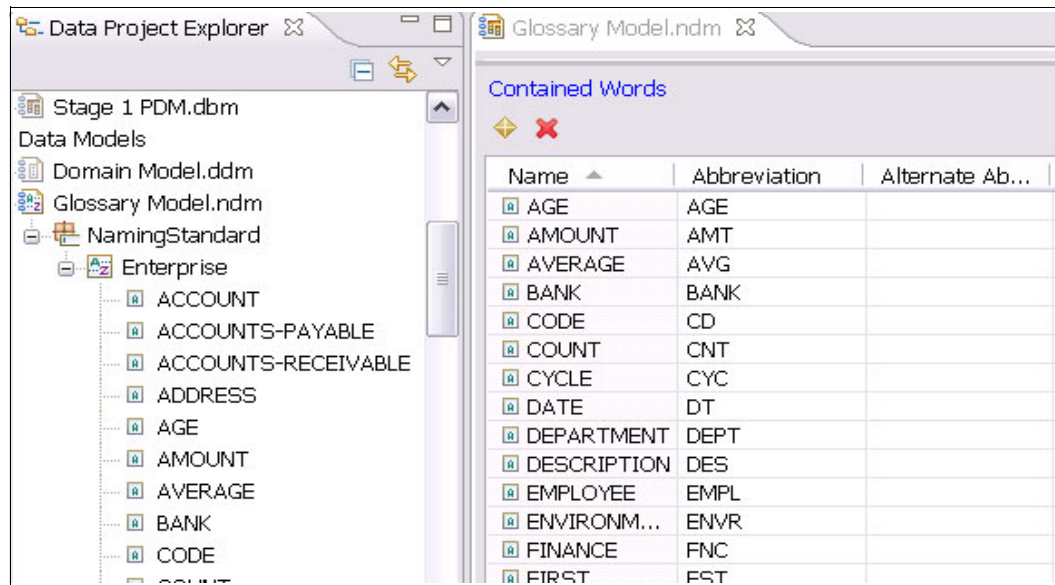


Figure 9-13 InfoSphere Data Architect Enterprise Glossary Model

Another feature of InfoSphere Data Architect is the capability to analyze an existing data model to ensure that it follows the naming conventions. The following subsections contain some naming suggestions for various database objects, such as entities (tables), constraints, and columns.

For more information about how to use the InfoSphere Data Architect Glossary Model, read the IBM developerWorks article at this website:

<http://www.ibm.com/developerworks/data/library/techarticle/dm-07011iu/>

LDM entity names

In the LDM, consider using more descriptive names for various entities (tables, constraints, and so on). A DB2 for i table name can be 128 characters in length. A physical file object name can be only 10 characters in length. Consider making the long table name meaningful. Use your standard abbreviation glossary model to create the short name, matching the long name as close as possible while staying within 10 characters.

Most graphical tools (InfoSphere Data Architect, System i Navigator, and so on) group objects by object type within a schema. It is not as important to include the application code or object type in the name. In addition, as you go forward, the physical file should never be referenced directly. All access is done through a view or stored procedure. Furthermore, common stored procedures are used for inserts, updates, and deletes.

Constraint names

Tools such as InfoSphere Data Architect can standardize constraint names that are generated when adding a constraint to a table. DB2 for i generates a constraint name if one does not exist in the **CREATE TABLE** or **ALTER TABLE** statements.

Column names

Consider changing existing column names in the new base tables. Remember in this phase that you do not want to break or modify existing programs. The surrogate logical files or views retain the original column names. New column names should be meaningful. The usage of abbreviations and acronyms must be defined in the glossary.

Adding the core fundamental items to the LDM entities

Primary/foreign keys, row change time stamps, and other missing core fundamental items are now added to the LDM tables.

Transforming the LDM to a Physical Data Model

When the core fundamental attributes are added to the LDM, the next step is to transform the LDM to a Physical Data Model (PDM). This is required to ensure that attributes were transformed correctly and to generate the new table DDL.

To transform the LDM to a PDM in InfoSphere Data Architect, complete the following steps:

1. Open the LDM you want to transform. In the Data Project Explorer, double-click the data model to open it.
2. From the main menu, click **Data** → **Transform** → **Physical Data Model**.
3. Complete the transformation wizard. Use the same name as the LDM for the PDM. The file extension differentiates the two.
4. Choose DB2 for i and the target release. If your release is not in the list, choose the highest release level.
5. Open the new PDM and validate that the table, column, and constraint names are correct.

Creating the future state PDM Using IBM Data Studio

In our example, we use InfoSphere Data Architect tool to create our PDM from an LDM. If you did not create the PDM from the LDM, copy the current state PDM to create a PDM representing the future state of the database. This model is the “future state PDM”. Expand the future state PDM and locate your schema (in our example, this is FLGHT400). Rename the schema to something meaningful. This is the schema that becomes the physical data layer.

Figure 9-14 shows our example of creating the future state PDM.

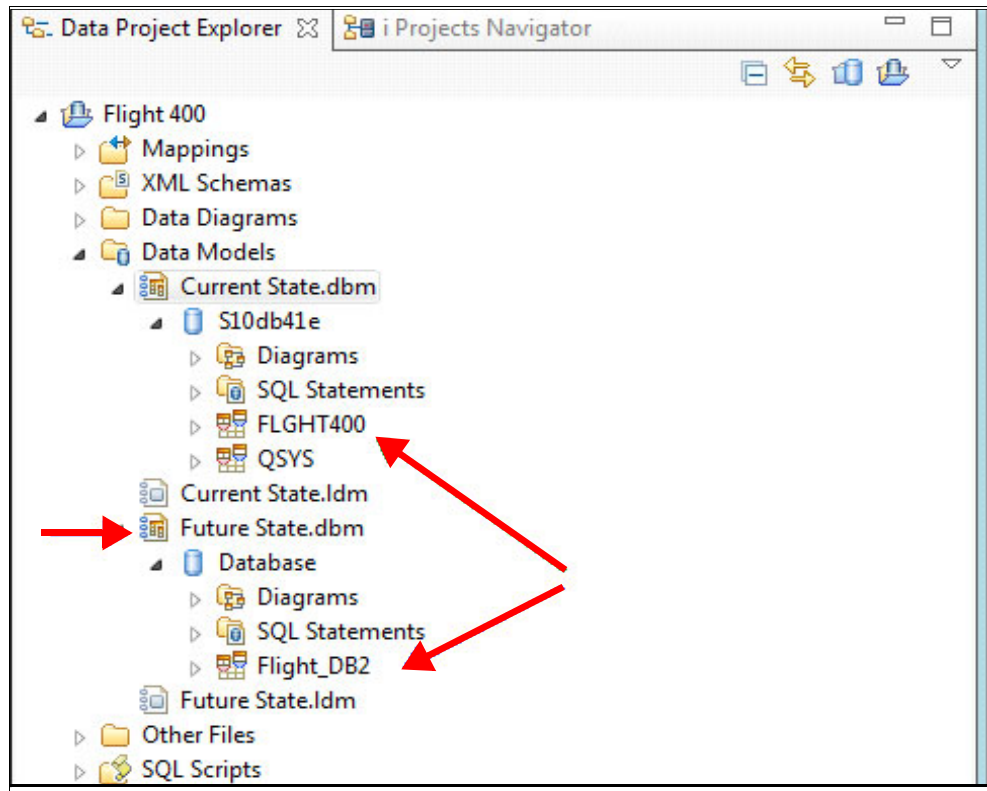


Figure 9-14 Example of creating the future state PDM

Note: Schema names can be up to 128 characters

Creating the DDL for the new SQL tables

When the PDM is validated, generate the SQL DDL that is required to create the tables. Here are the steps to complete this task:

1. Creating or enhancing the future state PDM
2. Generating DDL from the future state PDM
3. Deploying the tables
4. Maintaining the Field Reference File (optional)
5. Creating and deploying the surrogate file
6. Altering and deploying the DDS Logical Files
7. Migrating the data

Enhancing the future state PDM

Beginning with DB2 for i 6.1, you can now identify a time stamp column as a row change time stamp, which means that the time stamp column is implicitly updated with the default value (CURRENT_TIMESTAMP) whenever a row is added or changed within the table. In addition, DB2 for i 6.1 also provides hidden column capabilities. In essence, newly added columns can be omitted from the projected result set if a **SELECT *** is used to select all columns from a table.

During the Phase 0 discovery phase in our example, we find a DDS Join Logical File, which explicitly defines a relationship between the customer's table and the orders table. This is an excellent opportunity to prepare these two tables for Referential Integrity (RI).

Begin by expanding the future state schema (Flight_DB2 in our example). Right-click **Diagrams** and select **New Blank Diagram**. A blank diagram opens in the editor window. Find the General tab in the Properties view and rename the diagram to something meaningful (for example, Phase 1 Customer Orders). This step is shown in Figure 9-15.

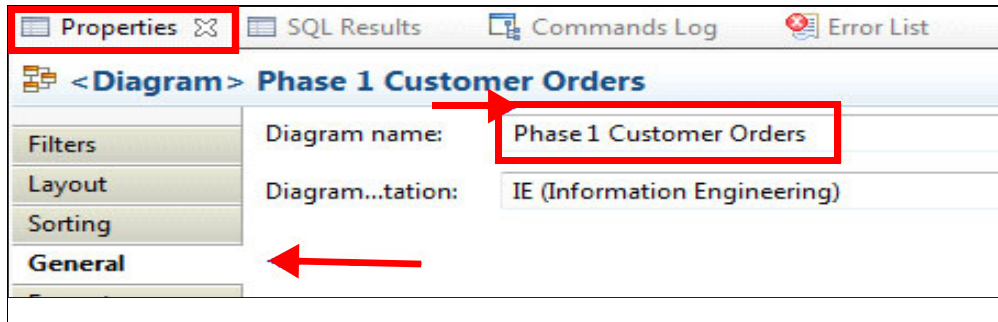


Figure 9-15 Updating the name of the diagram to a useful name

Drag the CUSTOMERS and ORDERS tables from the future state PDM to the blank diagram, as shown in Figure 9-16.

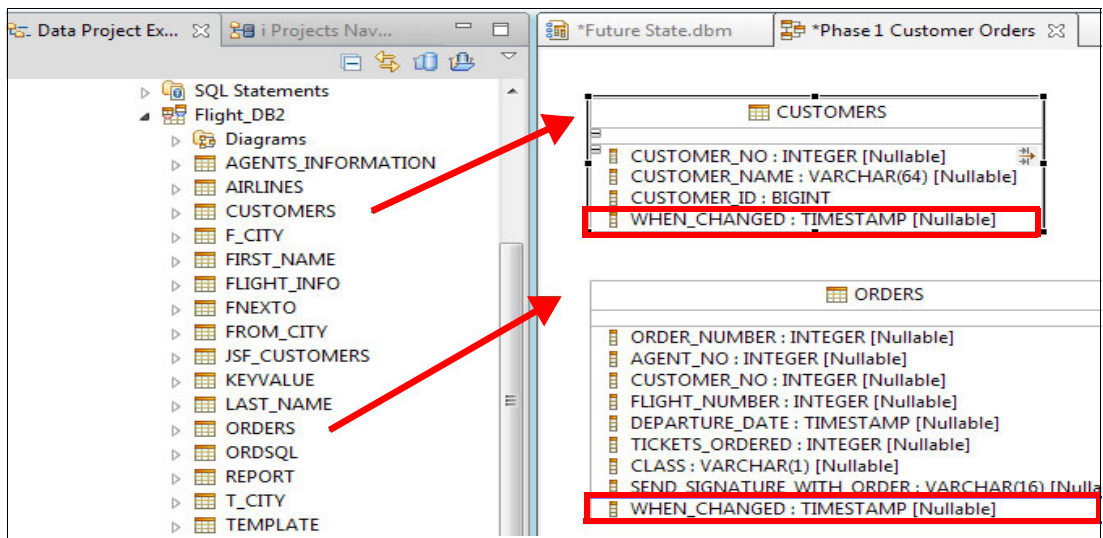


Figure 9-16 Dragging the CUSTOMERS and ORDERS tables from the future state PDM

Because we transformed the future state PDM from our future state LDM, the CUSTOMERS table already contains a new identity column. If you did not use an LDM, you can add an identity column now. In either case, you must review each new column in the diagram and add the IMPLICITLY HIDDEN and ROW CHANGE TIMESTAMP attributes as needed.

To add an identity column, click the table in the diagram (for example, CUSTOMERS) and select the **Columns** tab in the Properties view. Click the plus (+) symbol at the top of the column list to create a column entry. Modify the generated name to something meaningful (that is, CUSTOMER_ID). Define the column as a numeric type (an integer type such as BIGINT is an excellent choice). Select the check boxes **Primary Key and Not Null and Generated**. Select **Choose As Identity** from the Default Value/Generate Expression list.

To add a row, change the time stamp column to your tables and click the plus (+) symbol. Name the new column WHEN_CHANGED. Define the column as a TIMESTAMP. Select the **Not Null** check box.

Our diagram now looks similar to the one that is shown in Figure 9-17.

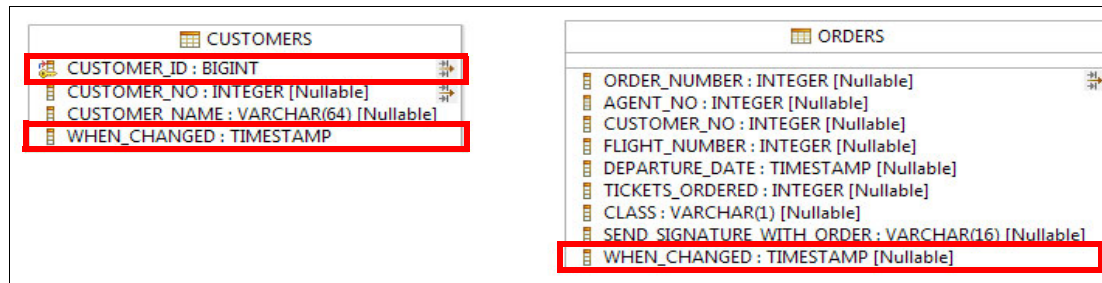


Figure 9-17 Updated diagram example

The CUSTOMER_ID column appears at the top of the columns in the diagram because it is designated as the primary key. Click the **CUSTOMER_ID** column and click the **Type** tab from the Properties view. Modify the Generate type to BY_DEFAULT. Scroll to the bottom of the Type view and select the **Hidden** check box, as shown in Figure 9-18.

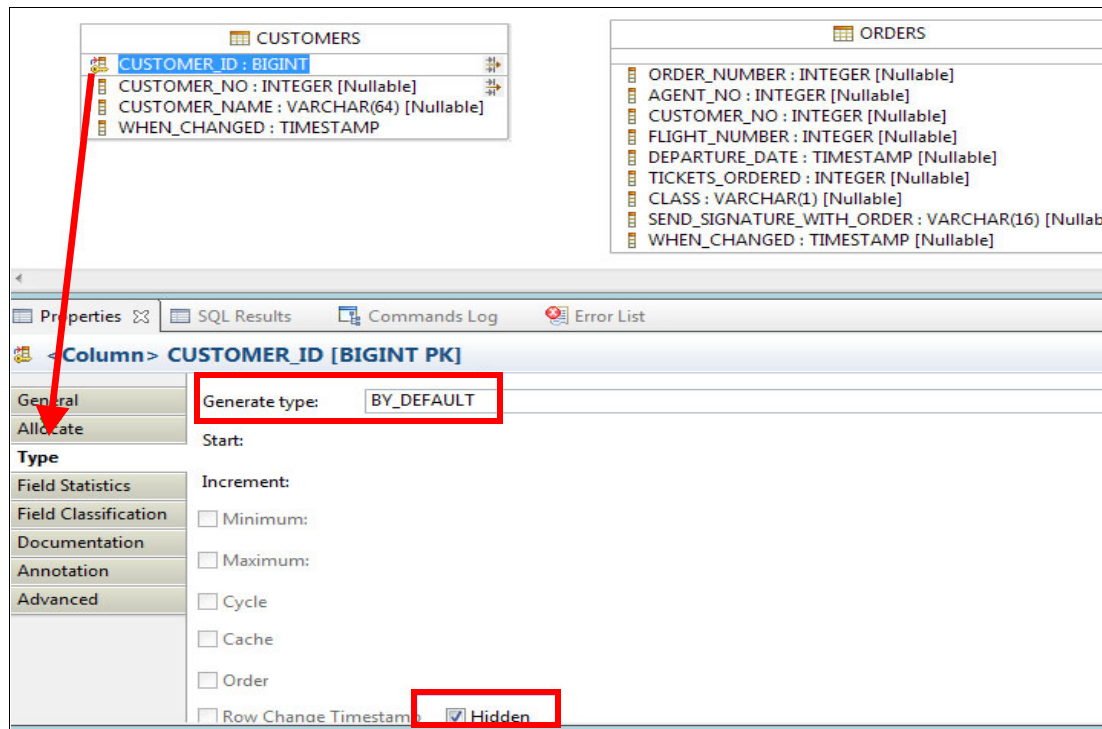


Figure 9-18 Adding or validating the Hidden attribute

Click each **WHEN_CHANGED** column per table and select the **Row Change Timestamp** and **Hidden** check boxes in the Type view.

If you did not use an LDM, now is the time to provide a new table name. Sometimes the object text provides a good long name for the table. In our example, we avoided spaces in the long name because this results in a quoted identifier. You must use quotation marks when referencing the name on SQL statements. We used the underscore character to separate words in the long name. An alternative approach is to use the short name in all SQL references of the table (for example, within an SQL view).

To change the table name, click the table and choose **General** from the Properties view. Change the Name field to the long name for the table. Enter a 10-character or less table short name in the System name field. This step is shown in Figure 9-19.

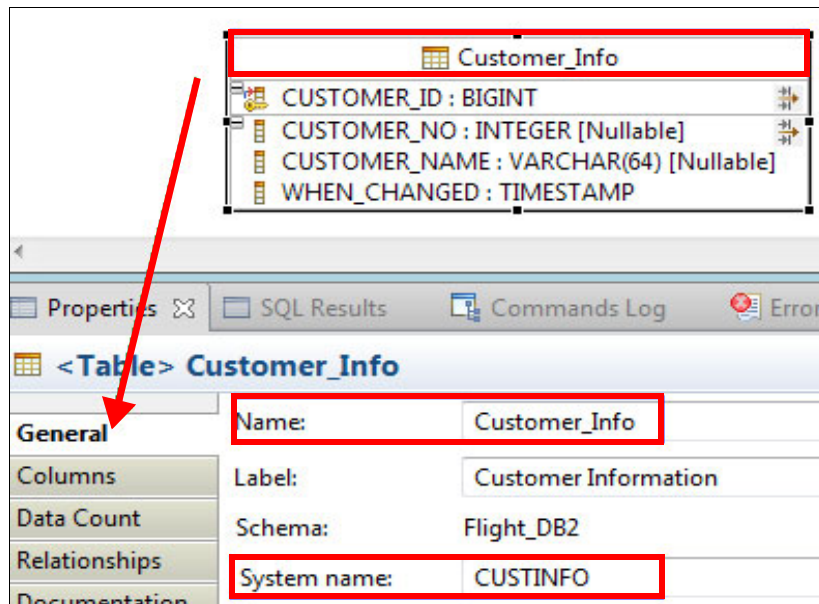


Figure 9-19 Updating the table field name

A short name is highly recommended and required if you have DDS LFs that are created over the new tables. If you have an existing naming convention for short names, you can use that. Otherwise, use standard abbreviations to construct a meaningful short name. The short name should not be the same as the current state physical file name.

Our Phase 1 Customer Orders diagram now looks like Figure 9-20.

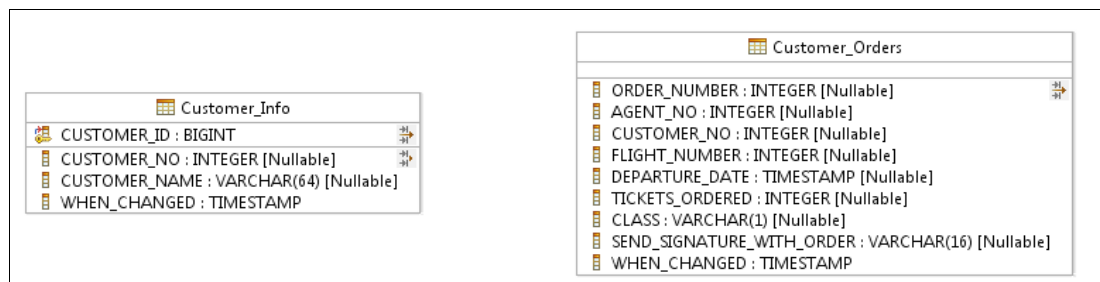


Figure 9-20 Updated Customer Orders diagram

The Flight/400 physical files are using some conventions that might not be typically found in most DDS generated databases. Optionally, you can choose to alter attributes now if you do not harm the existing applications. Here are some examples of attribute changes:

- ▶ Column name changes from short to long names
- ▶ Attribute changes, such as DECIMAL(9,0) to INTEGER, ZONED(9,2) to DECIMAL(15,2) or CHAR to VARCHAR
- ▶ DBCS to UNICODE
- ▶ Enhancing VARCHAR with ALLOCATE

Because we used the InfoSphere Data Architect tool, many of the implicit relationships were discovered for us. You might also infer an implicit relationship exists between ORDERS, AGENTS (AGENT_NO), and FLIGHTS (FLIGHT_NUMBER). As a preferred practice, start with a pair of tables and then refactor other related tables: however, you can add the foreign key columns now.

Now, the file unique key (for example, CUSTOMER_NO) is still used to establish application integrity between existing tables (for example, Customer_Info and Customer_Orders). Because the identity column is hidden, there should be no impact on existing applications. Also, because the identity column is the designated primary key, it is used for all future creation of customer-dependent tables, which includes restructuring current tables in phases 2 and 3.

Generating DDL from the PDM

In our example, we generated the DDL based on the Phase 1 Customer Orders diagram. To do so, click **Data** and then select **Generate DDL**. The Generate DDL Options window opens, as shown in Figure 9-21.

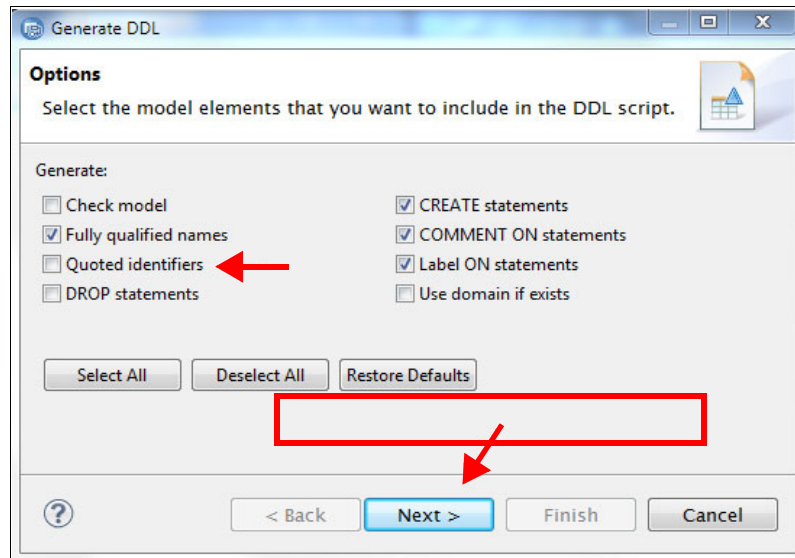


Figure 9-21 Generate DDL wizard

Clear the option for quoted identifiers (if you do not want spaces in your long names). Click **Next** in each subsequent window until the Save and Run window opens. Specify the same folder as the data design project, and give the DDL the same name as the diagram. In addition, select the **Open DDL file for editing** check box and click **Next**, as shown Figure 9-22 on page 379.

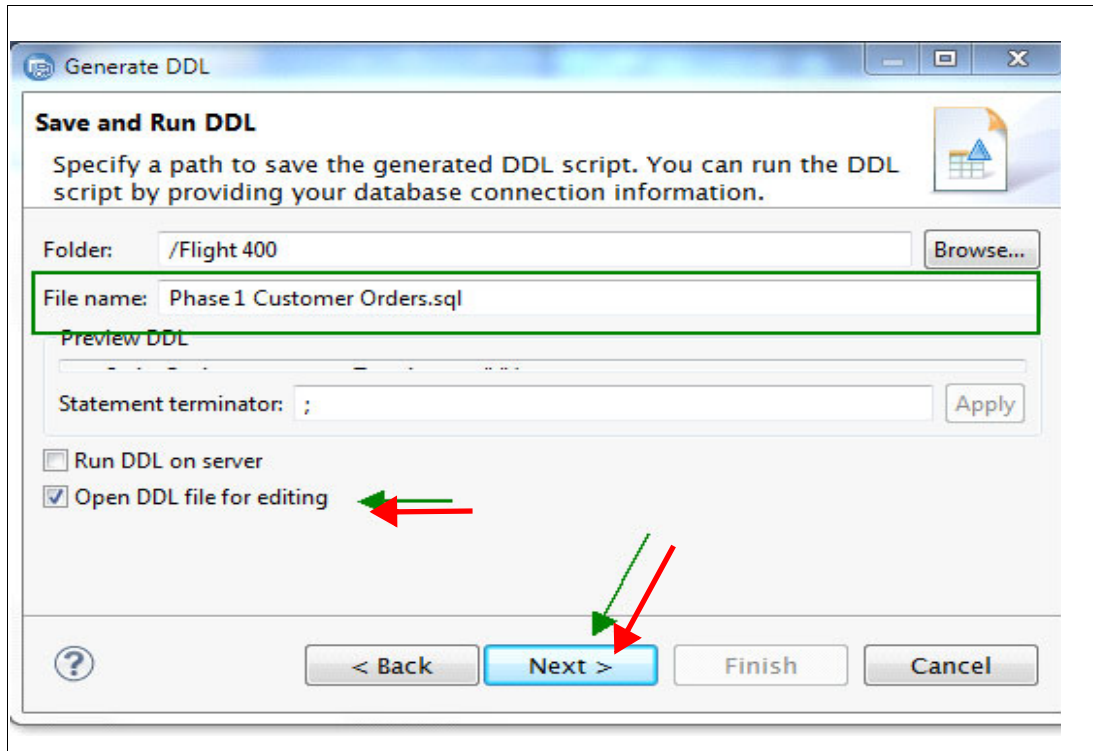


Figure 9-22 Specifying the same folder name as the data design object in the Generate DDL wizard

Click **Finish** to open a script window containing the DDL statements that were generated from the diagram. Review the generated script to ensure all columns, new columns, and labels (file and column text) are present. Example 9-7 shows the SQL DDL script that was generated from the Phase 1 Customer Orders Diagram (the added core fundamental items are in **bold** text).

Example 9-7 SQL DDL script that is generated from the Phase 1 Customer Orders Diagram

```
CREATE TABLE Flight_DB2.Customer_Info (
    CUSTOMER_NO INTEGER,
    CUSTOMER_NAME VARCHAR(64),
    CUSTOMER_ID BIGINT IMPLICITLY HIDDEN GENERATED BY DEFAULT AS IDENTITY ,
    WHEN_CHANGED TIMESTAMP NOT NULL IMPLICITLY HIDDEN GENERATED ALWAYS FOR EACH
ROW ON UPDATE AS ROW CHANGE TIMESTAMP
);

RENAME TABLE Flight_DB2.Customer_Info TO SYSTEM NAME CUSTINFO;

CREATE TABLE Flight_DB2.Customer_Orders (
    ORDER_NUMBER INTEGER,
    AGENT_NO INTEGER,
    CUSTOMER_NO INTEGER,
    FLIGHT_NUMBER INTEGER,
    DEPARTURE_DATE TIMESTAMP,
    TICKETS_ORDERED INTEGER,
    CLASS VARCHAR(1),
    SEND_SIGNATURE_WITH_ORDER VARCHAR(16),
    WHEN_CHANGED TIMESTAMP NOT NULL IMPLICITLY HIDDEN GENERATED ALWAYS FOR EACH
ROW ON UPDATE AS ROW CHANGE TIMESTAMP
)
```

```

);

CREATE UNIQUE INDEX Flight_DB2.CUSTINFO_IDX
  ON Flight_DB2.Customer_Infonull;

CREATE UNIQUE INDEX Flight_DB2.ORDERS_IDX
  ON Flight_DB2.Customer_Ordersnull;

ALTER TABLE Flight_DB2.Customer_Info ADD CONSTRAINT Flight_DB2.CUSTOMERS_PK
PRIMARY KEY (CUSTOMER_ID);

LABEL ON TABLE Flight_DB2.Customer_Info IS
'Customer Information';

LABEL ON TABLE Flight_DB2.Customer_Orders IS
'Customer Orders';

LABEL ON COLUMN Flight_DB2.Customer_Orders.SEND_SIGNATURE_WITH_ORDER IS
' SEND_SIGNATURE          WITH_ORDER';

```

Deploying the tables

In our example, we created the tables in a new, separate schema. Use this schema for all new tables, indexes, and constraints. The new tables are shared by many applications. The existing applications can remain in their current IBM i library.

As a preferred practice, export the DDL script to the IBM i IFS directory so that you can run the script in a batch job by using the Run SQL Statement (**RUNSQLSTM**) IBM i command. In addition, the **RUNSQLSTM** command can be scheduled to run during off peak hours.

In our example, we chose to run the SQL script directly from the SQL script window by clicking the green **Run** symbol, as shown in Figure 9-23.

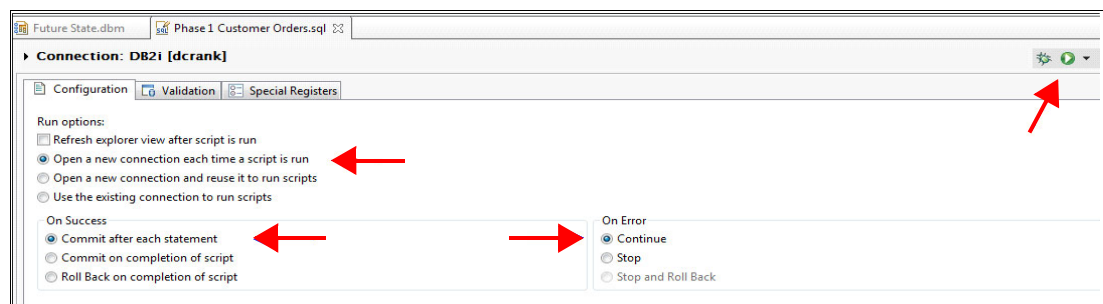


Figure 9-23 Running the SQL script from the SQL script window

By default, the SQL script runs in a new connection. We keep the default Commit after each statement and Continue on error default values. Using these default values prevents the entire script from being rolled back if an error occurs. Validate that the execution is successful by using the SQL Results view, as shown in Figure 9-24 on page 381.

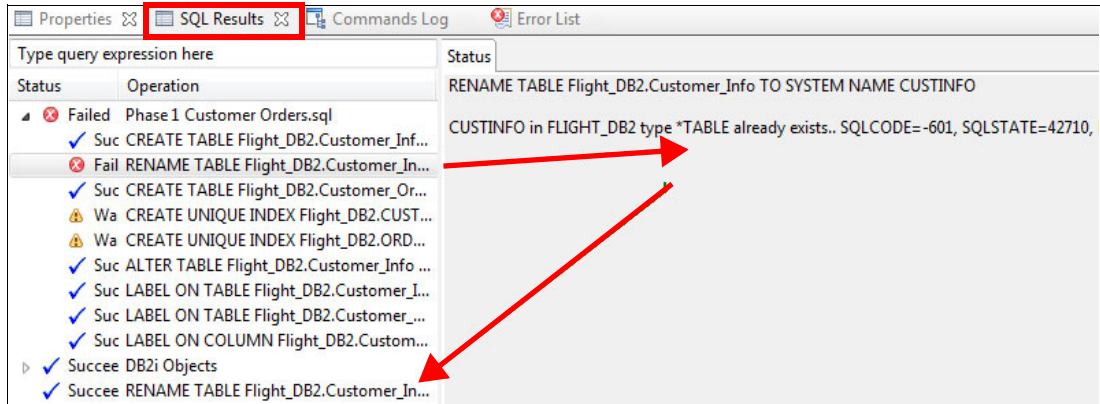


Figure 9-24 SQL run results window

In Figure 9-24, an error occurred with the **RENAME TABLE** statement because of the existence of a table with the same short name. After resolving the issue, select the single **RENAME TABLE** statement and run it by using the **Run** select statements option.

Use the Data Source Explorer view (Figure 9-25) to validate that the objects exist along with all attributes in the new schema (Figure 9-26).

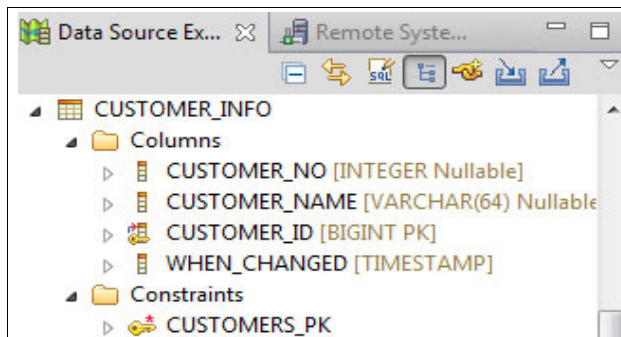


Figure 9-25 Verifying that a table object exists in the Data Source Explorer view

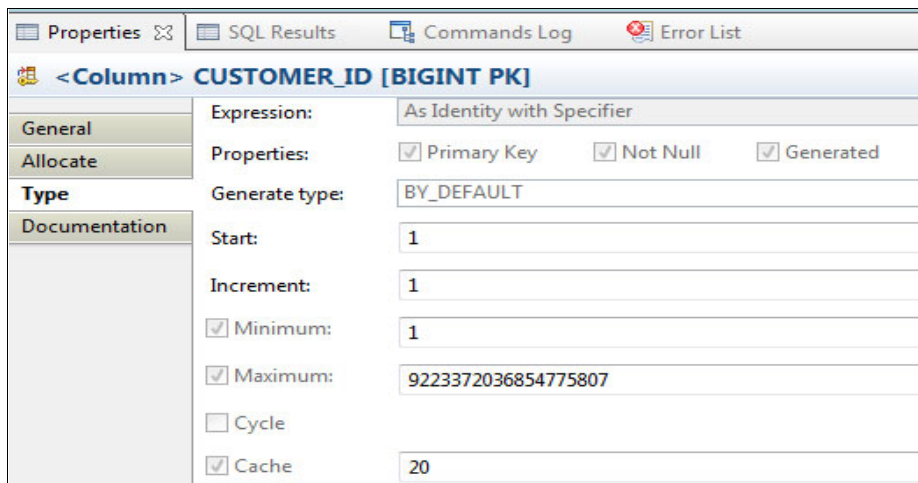


Figure 9-26 Verifying the table object attributes

Maintaining the Field Reference File

The Flight/400 DDS PFs did not use a Field Reference File (FRF). An FRF is intended to provide passive data dictionary support for the creation of DDS objects (database, display, and printer). The file level DDS keyword **REF** defines the FRF that is used to obtain the column attributes. The field level keyword **REFFLD** can be used to specify a column in the FRF that is used for the attributes.

Example 9-8 shows examples of using an FRF:

Example 9-8 Examples using a Field Reference File

The following fields are defined in DDS file FLDREFFILE

```
SUPPLYCOST 15P 2  EDTCDE(K $)
ADDRESS     30
```

The following is a DDS database file that references the above fields

```
                                REF FLDREFFILE
SUPPLYCOST  R
ADDRESS1    R      REFFLD(ADDRESS)
ADDRESS2    R      REFFLD(ADDRESS)
```

The following is a DDS display file that references SUPPLYCOST for its presentation attributes

```
                                REF FLDREFFILE
SUPPLYCOST  R
```

Because SQL DDL does not contain presentation attributes, it becomes important to ensure that DDS presentation keywords are not lost when the DDS PF is replaced with a DDL table. If you have an FRF, make sure that all fields that are referenced in display or printer files exist in the FRF. Also, check the display and printer file DDS for references to the original DDS PF. These items eventually must be changed to use the FRF; however, this task does not need to be done as part of this project.

If you are using REFFLD support but are not using an FRF, then consider creating one for your heritage DDS. Simply create a DDS PF without any members. Copy the DDS source for the DDS PF that is being re-engineered into the FRF source member. Do this for each DDS PF as it is being re-engineered.

If you are not using DDS field reference support (that is, REFFLD), an FRF is not required.

Creating and deploying the surrogate file

One of the most important goals of the DDS to DDL modernization strategy is to minimize the impact of change. The main objective is the successful migration of the data from an existing DDS created database to a DDL created database without impacting the existing programs. This process is accomplished by ensuring that the existing Record Format Level ID (RFID) of the file to be re-engineered is not altered.

The IBM DDS to DDL modernization strategy uses a surrogate file, which retains the original format. This process allows the missing core fundamental items to be added to the new table, without forcing program changes or recompilation. (Some Source Change Management products force recompilation during the deployment of the table.) The surrogate file can be one of the following things:

- ▶ A DDS Logical File that contains the columns that are defined in the original DDS physical file record format. This file can be an existing DDS LF or the original DDS PF converted to a DDS LF.

- ▶ A DDS Physical File, which is a replica of the original DDS PF but does not contain data. It is used for record format reference only.
- ▶ An SQL INDEX that is used for keyed physical and logical files, which are being transformed to non-partitioned indexes built over local partitioned tables.

In addition, many customers have been using DDL-based databases for some time. A DDL defined database does not mean that it is exempt from reengineering, especially if it does not conform to good relational database design standards. The modernization strategy works the same. Here are examples of SQL surrogate tables:

- ▶ An SQL VIEW, which is named the same as the original base table that contains the original format.
- ▶ An SQL ALIAS, which is named the same as the original SQL base table but is defined on the new SQL base table.
- ▶ An SQL INDEX, which is used for keyed physical and logical files that are being transformed to non-partitioned indexes built over local partitioned tables.

The existing programs that currently reference the DDS physical and logical files are unaware of the transformation and continue to work as intended. Because the new columns are not part of the DDS LF, all new rows that are inserted through the RPG WRITE operation automatically result in default values being used for the identity column, data change columns, and session user.

Altering and deploying DDS logical files

In our example, we modified all existing DDS logical files that reference the DDS PFs as follows:

1. We changed the **PFILE** keyword to the system name (short name) of the new table.
2. Optionally, the **FORMAT** keyword must be added to the DDS LF if it references the format of the original physical file. The Flight/400 DDS LFs all used unique formats.

Figure 9-27 contains a table that shows the before and after images of a transformed DDS PF to DDL table. In our case, this file is the AGENTS file (these images were captured from the current and future state physical data models).


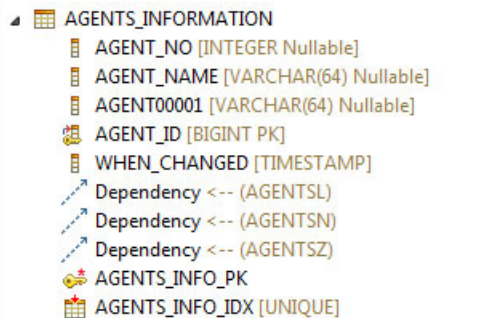
Original Structure of AGENTS with dependencies	New Structure of AGENTS_INFORMATION with dependencies
 <pre> AGENT_NO [INTEGER Nullable] AGENT_NAME [VARCHAR(64) Nullable] AGENT00001 [VARCHAR(64) Nullable] Dependency <-- (AGENTS_L) Dependency <-- (AGENTS_N) Dependency <-- (AGENTS_Z) AGENTS [UNIQUE] </pre>	 <pre> AGENT_NO [INTEGER Nullable] AGENT_NAME [VARCHAR(64) Nullable] AGENT00001 [VARCHAR(64) Nullable] AGENT_ID [BIGINT PK] WHEN_CHANGED [TIMESTAMP] Dependency <-- (AGENTS_L) Dependency <-- (AGENTS_N) Dependency <-- (AGENTS_Z) AGENTS_INFO_PK AGENTS_INFO_IDX [UNIQUE] </pre>

Figure 9-27 Agents file before and after

Figure 9-28 contains a table that shows the before and after images of the AGENTS physical file.


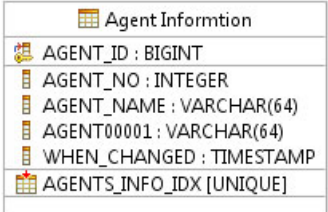
Original Structure of AGENTS PF	New Structure of AGENTS_INFORMATION (New DDL SQL Table)
	

Figure 9-28 Agents physical file before and after

Figure 9-29 on page 385 contains a table that shows the before and after DDS for the existing logical files that are built over the agents table. In all cases, the **FORMAT** keyword was not required.

Original DDS source built over AGENTS	New DDS source built over AGENTS_INFORMATION
Physical file AGENTS	Logical file AGENTS (surrogate file)
<pre> R ORDERS UNIQUE AGENT_NO 9B 0 ALWNULL AGENT_NAME 64A ALWNULL CCSID(37) VARLEN AGENT00001 64A ALWNULL COLHDG('AGENT_PASSWORD') CCSID(37) VARLEN K AGENT_NO </pre>	<pre> R ORDERS UNIQUE AGENT_NO 9B 0 ALWNULL AGENT_NAME 9B 0 ALWNULL AGENT_NAME 64A ALWNULL AGENT_NAME 64A ALWNULL CCSID(37) VARLEN AGENT00001 64A ALWNULL AGENT00001 64A ALWNULL COLHDG('AGENT_PASSWORD') CCSID(37) VARLEN K AGENT_NO </pre>
Logical file AGENTSL	
<pre> R ANAMER PFILE(AGENTS) AGNMBR R RENAME(AGENT_NO) AGNAME R RENAME(AGENT_NAME) AGPASS R RENAME(AGENT00001) K AGNAME </pre>	<pre> R ANAMER PFILE(AGENTSINFO) AGNMBR R RENAME(AGENT_NO) AGNAME R RENAME(AGENT_NAME) AGPASS R RENAME(AGENT00001) K AGNAME </pre>
Logical file AGENTSN	
<pre> R ANAMER PFILE(AGENTS) AGNMBR R RENAME(AGENT_NO) AGNAME R RENAME(AGENT_NAME) AGPASS R RENAME(AGENT00001) K AGNMBR </pre>	<pre> R ANAMER PFILE(AGENTSINFO) AGNMBR R RENAME(AGENT_NO) AGNAME R RENAME(AGENT_NAME) AGPASS R RENAME(AGENT00001) K AGNMBR </pre>
Logical file AGENTSZ	
<pre> R AGNTRZ PFILE(AGENTS) AGNT#Z 9S 0 RENAME(AGENT_NO) AGNTNZ RENAME(AGENT_NAME) AGNTPZ RENAME(AGENT00001) K AGNT#Z </pre>	<pre> R AGNTRZ PFILE(AGENTSINFO) AGNT#Z 9S 0 RENAME(AGENT_NO) AGNTNZ RENAME(AGENT_NAME) AGNTPZ RENAME(AGENT00001) K AGNT#Z </pre>

Figure 9-29 Before and after for DDS for existing logical files

In Figure 9-29, the DDS LFs were originally built over a physical file. The PFILE now specifies the new SQL table system name. The format for the new table is irrelevant moving forward; however, existing programs still reference the DDS PF format. In our case, the logical files each contained their own set of columns.

Many logical files are created by using a technique that is referred to as format sharing. In this case, the fields that are defined in the physical file are implicitly added to the logical file format. For these situations, the DDS **FORMAT** keyword allows the DDS LF to find the old format even though it is built over the new table. Programs referencing the DDS LF do not need to be changed or recompiled.

Figure 9-30 shows a table with the before and after for the DDS logical file using the **FORMAT** keyword.

Example of DDS LF using format sharing	New DDS source
R ORDERS PFILE (AGENTS) K AGENT_NO	R ORDERS PFILE (AGENTSINFO) FORMAT (AGENTS) K AGENT_NO

Figure 9-30 Before and after for the DDS logical file using the **FORMAT** keyword

The following steps outline an alternative method when reengineering DDS PFs to DB2 for IBM i local partitioned tables:

1. Reverse-engineer the DDS keyed LF, built over the DDS PF, to an SQL INDEX. Ensure that the **RCDFMT** keyword is present and matches the format of the LF.
2. Add the **NOT PARTITIONED** clause to the **CREATE INDEX** statement.

Note: This method is primarily a bridge for those customers who are close to exceeding the current size limits of a single member file currently being accessed by traditional I/O methods. The surrogate INDEX provides continued access while the data access layer is created. When complete, all access to the partitioned table is by SQL. The non-partitioned surrogate INDEX can then be dropped or re-created as a partitioned INDEX.

In our example, we ran the following SQL script (Example 9-9) in InfoSphere Data Architect to verify the old and new format IBM Data Studio. The IBM Data Studio or System i Navigator Run SQL Scripts tools can also be used.

Example 9-9 Example SQL script

```
CALL qsys2.qcmdexc (
'DSPFD FILE (FLGHT400/AGENTS*) TYPE(*RCDFMT) OUTPUT(*OUTFILE)
OUTFILE(QTEMP/FORMATS)');

CALL qsys2.qcmdexc (
'DSPFD FILE (FLGHT400M/AGENTS*) TYPE(*RCDFMT) OUTPUT(*OUTFILE)
OUTFILE(QTEMP/FORMATS) OUTMBR(*FIRST *ADD)');

select rffile, rflib, rffila, rfname, rfid from qtemp.formats
order by rfname, rffile;
```

Figure 9-31 on page 387 shows the results from the script in Example 9-9.

File	Library	File Attribute	Record Format	Format Level Identifier
AGENTSZ	FLGHT400	*LGL	AGNTRZ	2DF03658A36B4
AGENTSZ	FLGHT400M	*LGL	AGNTRZ	2DF03658A36B4
AGENTSZ	FLGHT400	*LGL	ANAMER	2DADA3D93D378
AGENTSZ	FLGHT400M	*LGL	ANAMER	2DADA3D93D378
AGENTSZ	FLGHT400	*LGL	ANAMER	2DADA3D93D378
AGENTSZ	FLGHT400M	*LGL	ANAMER	2DADA3D93D378
AGENTS	FLGHT400	*PHY	ORDERS	2B0AEBE8D076F
AGENTS	FLGHT400M	*LGL	ORDERS	2B0AEBE8D076F

Figure 9-31 Example SQL script output results

Many changes can be made to the new table column attributes without requiring changes or recompilation to the existing application programs.

Migrating the data

When the new tables, indexes, existing SQL views, and traditional DDS LFs are created, you can begin migrating the old data to the new tables. Ensure that all columns that are defined as numeric truly contained numeric data. DB2 for i does not allow invalid numeric data to be inserted in a column that is defined as numeric. This was not the case with DDS.

Place the new database objects into production at the completion of this step. To migrate the data, complete the following steps:

1. Validate numeric data. Correct any bad data that is found in the production files.
2. Back up the existing database objects from their current library.
3. Migrate the data from the DDS PF to the new table.

Example 9-10 shows a stored procedure that we used in our example to migrate the data from the old files to the new tables.

Example 9-10 SQL example

```
CREATE OR REPLACE PROCEDURE Migrate_Data (
  p_From_Schema VARCHAR(128),
  p_To_Schema VARCHAR(128))
  RESULT SETS 1
  LANGUAGE SQL
P1: BEGIN
  -- Variables
  DECLARE v_SQL_String CLOB;
  -- Declare cursor
  DECLARE cursor1 CURSOR WITH RETURN FOR
  With from_tables AS (
    SELECT TABLE_SCHEMA, TABLE_NAME, number_rows from_nbr_rows
    FROM QSYS2.SYSTABLES
    JOIN QSYS2.SYSTABLESTAT USING(table_schema, table_name)
    WHERE table_type = 'T' AND table_schema = USER)
  SELECT TABLE_SCHEMA, TABLE_NAME, from_nbr_rows, number_rows to_nbr_rows
  FROM from_tables
  JOIN QSYS2.SYSTABLESTAT USING(table_schema, table_name)
  ORDER BY table_name;
  -- Drop existing referential constraints
```

```

FOR FK1 AS
    Fk1 CURSOR FOR
    select table_name, constraint_name
    from qsys2.syscst
    where constraint_schema = p_To_Schema
    and constraint_type = 'FOREIGN KEY'
    ORDER BY table_name
    DO
        SET v_SQL_String = 'ALTER TABLE ' || RTRIM(p_To_Schema) || '.' ||
            RTRIM(table_name) || ' DROP CONSTRAINT ' ||
            RTRIM(p_To_Schema) || '.' ||
RTRIM(constraint_name);
        EXECUTE IMMEDIATE v_SQL_String;
    END FOR;
COMMIT;
-- Loop and insert from old to new
FOR V1 AS
    c1 CURSOR FOR
    SELECT table_name
    FROM QSYS2.SYSTABLES JOIN QSYS2.SYSTABLESTAT
    USING(table_schema, table_name)
    WHERE table_schema = p_From_Schema
    AND table_type = 'T'
    AND number_rows > 0
    ORDER BY table_name
    DO
        SET v_SQL_String = 'INSERT INTO ' || RTRIM(p_To_Schema) || '.' ||
            RTRIM(table_name) || ' SELECT * FROM ' ||
            RTRIM(p_From_Schema) || '.' || RTRIM(table_name);
        EXECUTE IMMEDIATE v_SQL_String;
    END FOR;
commit;

-- Display the result set
OPEN cursor1;
END P1

```

4. Validate that the migration was successful through queries against key business metrics. For example, besides equal active row counts and distinct unique key values, consider summing important quantity and dollar amount columns and comparing totals between the old and new objects.
5. If the validation is correct, replace the existing DDS PF and LFs in their current libraries with the modified DDS LFs (surrogate file and related LFs).
6. If the format IDs did not change, existing programs run without recompiling.
7. At the completion, you have new tables without deleted rows isolated to a separate schema.

Use the following techniques to improve the performance of the data migration:

- ▶ Use commitment control.
- ▶ Use SMP.
- ▶ Consider using a commitment control notify object.
- ▶ Use SQL Multi-row INSERT capabilities.
- ▶ Use a large Logical Unit of Work (LUW).

Using commitment control

The journal capability works concurrently with insert/update/delete events that occur against a journaled physical file. This asynchronous behavior assumes that the insert/update/delete activity is running under commitment control. The usage of commitment control is optional with DB2 for i.

When commitment control is not used, the asynchronous nature of journal is disabled. That actual insert/update/delete event is placed in a wait state until the journal receiver entry/entries are recorded to the auxiliary storage device. In essence, the journal receiver writes become synchronous, possibly resulting in excessive run times.

When commitment control is active the journal receiver entries are bundled together with other journal entries and written in one I/O operation to the journal receiver. This allows the insert/update/delete operation to no longer have to wait for the journal operation to complete.

Using SMP

There are two reasons behind this recommendation. The first is to take advantage of blocked inserts. A commit after each write flushes the buffer, essentially resulting in a blocking factor of one row. The second is to take advantage of SMP parallel index maintenance.

When SMP is enabled, each index can be maintained concurrently in its own thread. At insert time, the buffer is checked to determine the number of rows that are being written. The current minimum is eight rows to allow multiple index maintenance threads. The usage of a commit operation following each write operation results in one row in the buffer.

To meet the minimum, you can add a counter and delay the commit until the counter reaches at least 8. Do the commit and reset the counter to zero. The system sees at least eight rows in the buffer and creates multiple index maintenance threads.

Using a commitment control notify object

An additional benefit to using commitment control is the ability to restart at the last successful commit operation. This is accomplished by using a commit *notify object*. A notify object is a message queue, data area, or database file that contains information that identifies the last successful transaction that is completed for a particular commitment definition if that commitment definition did not end normally.

When a program is started after an abnormal end, it can look for an entry in the notify object. If the entry exists, the program can start a transaction again. After the transaction is started again, the notify object is cleared by the program to prevent it from starting the same transaction yet another time.

For a batch application, the commit identification can be placed in a data area that contains totals, switch settings, and other status information that is necessary to start the application again. When the application is started, it accesses the data area and verifies the values that are stored there. If the application ends normally, the data area is set up for the next run.

You can find more detailed information about notify objects and other commitment control concepts in the IBM i Information Center at this website:

<http://publib.boulder.ibm.com/infocenter/iseriess/v6r1m0/index.jsp>

Using SQL multi-row INSERT

Additional performance gains can be realized by reducing the calls to the IBM i QDB system modules. This is done by taking advantage of SQL blocked **INSERT** support.

SQL extends traditional blocking techniques by using multi-row **INSERT** (also known as application blocked **INSERT**). This technique eliminates the need to use an **OVRDBF** command because the normal behavior for SQL is to automatically block writes.

To take advantage of application blocking, you must add an externally described array or structure to your application program. Use the **OCCURS** or **DIM** keywords to create a multiple occurrence structure up to 32,767 (or 16 MB, whichever is less).

Using the program counter technique that was described previously, move the next record to be written to the host array or structure instead of performing an **INSERT**. Accumulate the maximum number of rows in the structure and then perform an **INSERT FOR n ROWS**, where n is equal to the program counter (either 32,767 or less). In essence, there is one MI call for n rows. DB2 for i determines the appropriate system blocking factor by using the most current optimal factors for the release you are on. When the **INSERT** completes, perform a **COMMIT** operation. Example 9-11 shows this technique in a code example.

Example 9-11 SQL multi-row INSERT example

```
D mySQLrowsDS    e DS                                extname(view-a) inz
D mySQLrowsMDS   DS                                likeDS(mySQLrowsDS)
D
D               occurs(32767)
D NRows          s                                5i 0 inz(32767)
```

```
exec sql declare cursor_1 scroll cursor for
       select * from view-a for fetch only;
exec sql open cursor_1;
exec sql fetch first from cursor_1 for :NRows rows into
       :mySQLrowsMDS;
```

```
dow sqlcod = 0;
  exec sql
    insert into table-a
      :NRows rows values (:mySQLrowsMDS);
  if sqlCod <> *ZERO and
     sqlCod <> 100;
     do_some_error_trapping.....
     exec sql close cursor_1;
     *inLR = *On;
     return;
  endif;
clear mySQLrowsMDS;
exec sql fetch next from cursor_1 for :NRows rows into
       :mySQLrowsMDS;
```

```
Enddo;
Exec sql close cursor_1;
```

.....

DB2 for i also provides a means in which to partially insert all rows up to a row in error. This is done by using the SQL **GET DIAGNOSTICS DB2_ROW_NUMBER** statement information item to return the occurrence of the row on the host structure that caused the error. The rows on the host structure up to the failing row can be reinserted (after the **ROLLBACK** operation). The notify object can indicate where to restart after the error condition is corrected.

Using a larger logical unit of work for inserting rows

This technique is not necessary if you are using SQL multi-row **INSERT** support.

Traditional I/O storage management has always attempted to block inserts for files that are designated as output only. In this case, the compiler generates a call to the QDBPUTM IBM i OS module. This module uses a default 8 K buffer to hold the pseudo-written records. When the buffer is filled, the records are written to disk using the *INSDSEN MI-Complex instruction.

The file that is being inserted into must be open for output only. The system might turn off blocking during the open of the output only file. This process is done to allow the application program to handle any errors that might occur during the write operation.

Controlling the blocking size is important for data-intensive, performance-critical applications:

- ▶ The Override with Data Base File (**OVRDBF**) command can be used to tune sequential read-only and write-only applications.
- ▶ A specific byte count can be supplied, or *BUF32KB, *BUF64KB, *BUF128KB, and *BUF256KB special values can be specified.

The new **OVERRIDE_TABLE()** procedure is an easy approach for SQL applications to control blocking programmatically, as shown in Example 9-12.

Example 9-12 OVERRIDE_TABLE procedure example

```
-- Override the Employee table to utilize 256K blocking for sequential processing
CALL QSYS2.OVERRIDE_TABLE('CORPDATA', 'EMP', '*BUF256KB');

-- Discard the override
CALL QSYS2.OVERRIDE_TABLE('CORPDATA', 'EMP', 0);
```

The system does not perform the insert until the buffer is full. Until that time, the records are not available to other applications. For the data migration, this should not be an issue.

If an error occurs, all entries in the buffer that are not written are lost.

CCSID and Unicode conversion

CCSID is the value that identifies the character set and code page for data in the database. At the DDS level, fields that contain character data (types A and O) can specify a CCSID explicitly or let the CCSID use default values that are based on the national language configuration of the system. The latter is more typical. Character fields that are designated with type A are tagged with a single-byte CCSID and O fields are tagged with a mixed CCSID encoding that allows both single and double byte representations with shift-in/shift-out characters.

Unicode is the industry solution for storing and managing data across a broad variety of national languages. IBM i supports UTF-8, UCS-2, and UTF-16 encoding. UTF-16 is recommended because it allows over a million different characters to be encoded, typically storing character data in a two-byte-per-character format.

Note: Using the surrogate file approach works with transforming DBCS fields to Unicode.

9.3.3 Phase 2 and 3

The objective of these phases is to create a virtual or data abstract layer. This layer is created by decoupling the HLL program database access code from the presentation and business program logic. Concurrently, duplicate columns across parent-child tables can be eliminated. This process includes the following tasks:

1. Creating SQL views of the database
2. Creating SQL I/O modules that contain SQL procedures to access the SQL views
3. Creating a bridge (handler) program to intercept the HLL program I/O operations and run the appropriate SQL service program I/O function

9.3.4 Creating SQL views

The first step to decoupling I/O is to ensure that an SQL view exists over the new SQL table that was created in Phase 1. All database enhancements should be done using the future state PDM.

The main goals of Phase 2 are shown in Figure 9-32.

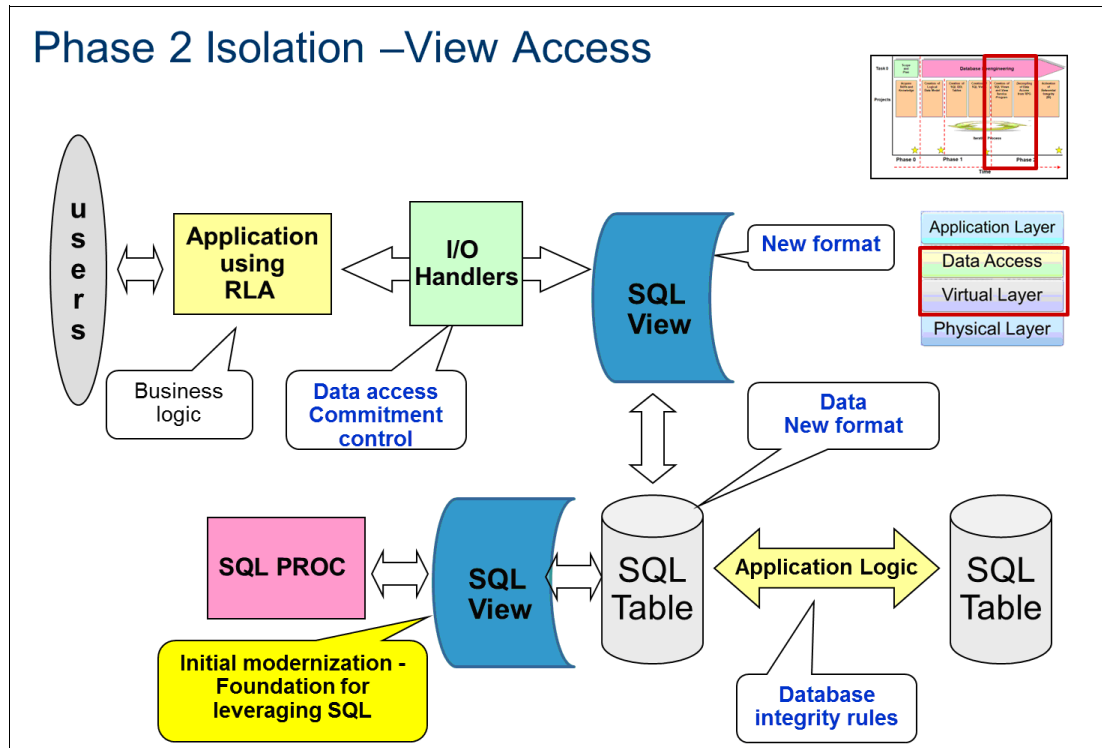


Figure 9-32 Phase 2 diagram

Start by creating a new Data Model diagram by copying the Phase 1 Customer Orders diagram that is shown in Figure 9-20 on page 377. The new diagram has a default name of Copy of Phase 1 Customer Orders diagram. Open the new diagram, click **Properties** → **General** and change the name to Phase 2 Customer Orders with Views.

The diagram should now look like Figure 9-33 on page 393.

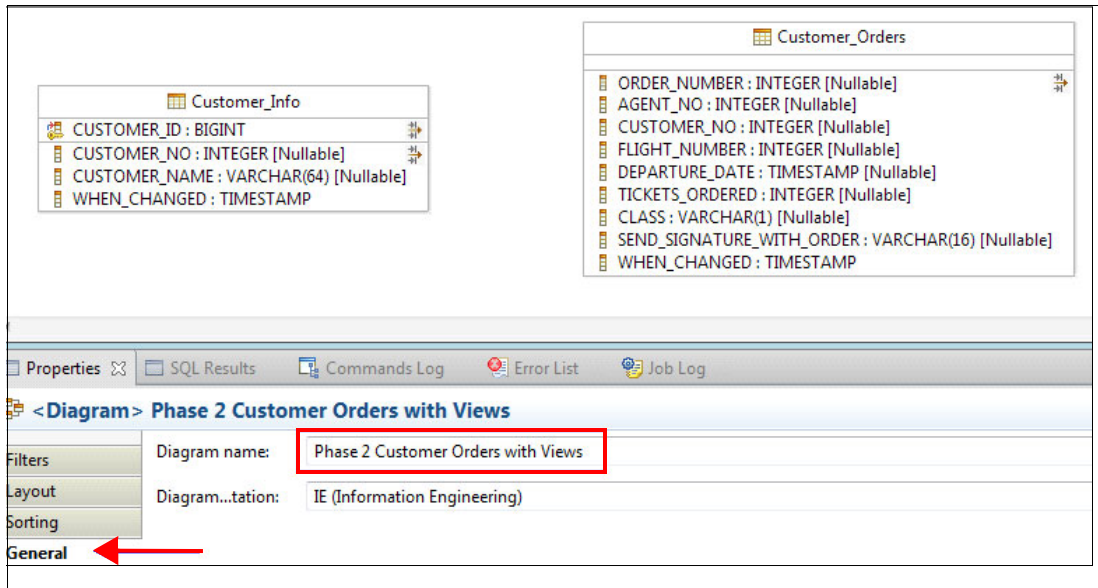


Figure 9-33 New Phase 2 Data Model diagram

In our example, we used the Palette tool to create two views in the diagram. We named the first view `Customer_Names` and the second view `Customer_Order`. We used `CUSTNAMES` as the System Name for the `Customer_Name` view and `CUSTORDER` as the system name for the `Customer_Order` view.

The diagram now looks like Figure 9-34.

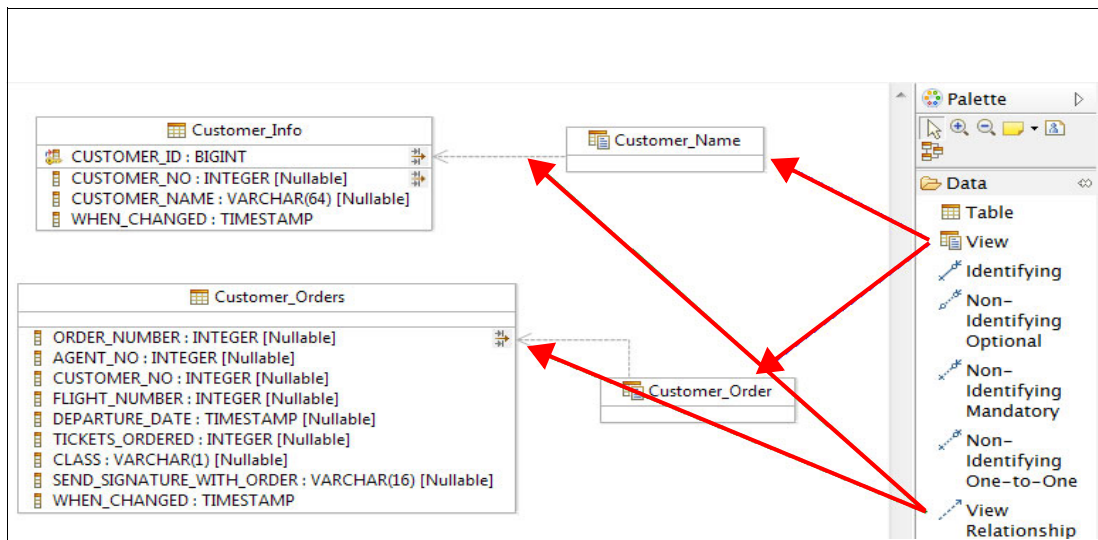


Figure 9-34 New `Customer_Name` and `Customer_Order` views added the diagram

The columns for the views are determined by the SQL `SELECT` statement that is contained in the view. To add the SQL statement, click the **SQL** tab in the Properties view for the `Customer_Name` view and enter the following SQL statement:

```
SELECT * FROM Customer_Info
```

Click **Update** to validate the statement and update the diagram.

Repeat this process for the Customer_Order view and enter the following SQL statement:

```
SELECT * FROM Customer_Orders
```

The Columns tab and the views in the diagram now contain the columns in the projected result set.

Change the layout of the diagram by clicking any open space in the diagram and then clicking the **Layout** tab (as shown in Figure 9-35) in the diagram Properties view.

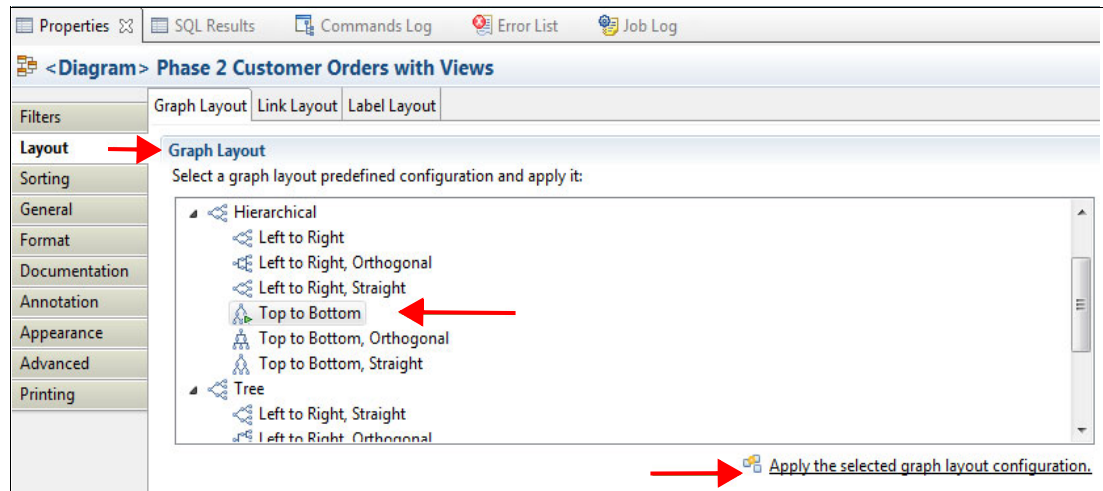


Figure 9-35 Changing the layout of the Customers Order View

Our diagram now contains the completed views, as shown in Figure 9-36. The views Customer_Names and Customer_Order do not contain the hidden columns CUSTOMER_ID or WHEN_CHANGED. These views are used by the data access modules, which perform the **INSERT**, **UPDATE**, and **DELETE** operations. If your view is missing, you must add the hidden fields.

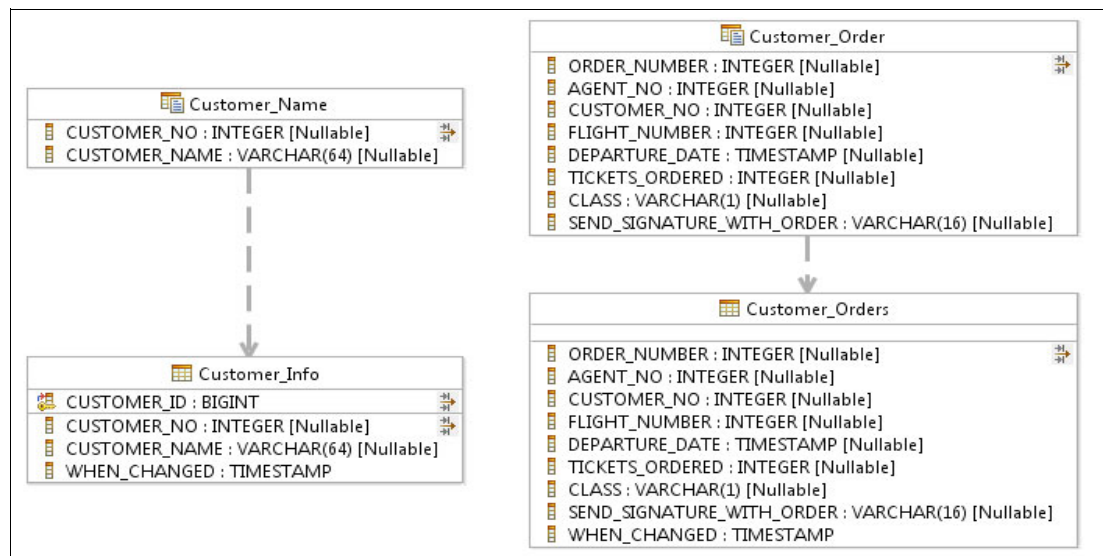


Figure 9-36 Completed views

Generate the DDL for the views by highlighting each view in the diagram and clicking **Data** → **Generate DDL** from the menu bar. Again, clear the **Quoted Identifiers** option and click **Next** twice to continue to the Save and Run DDL window. Enter “Phase 2 Customer Orders with Views” in to the File name field and then select the **Open DDL file for editing** check box. Click **Next** and then **Finish**.

In our example, we modified the SQL script by replacing all references to FLIGHT_DB2 with FLGHT40M (the name of our application schema). We added the following SQL statement to the beginning of the script (the name of the schema is the future state schema, in this case FLIGHT_DB2):

```
SET CURRENT SCHEMA FLIGHT_DB2;
```

By default, DB2 for i searches the current schema for all unqualified table references. The SQL statement that we coded within the views did not qualify the base tables. We also inserted the **OR REPLACE** between **CREATE** and **VIEW**. Example 9-13 shows the edited SQL script.

Example 9-13 Edited SQL script

```
SET CURRENT SCHEMA FLIGHT_DB2;

CREATE OR REPLACE VIEW FLGHT400.Customer_Names (CUSTOMER_NO, CUSTOMER_NAME) AS
select * from customer_info;

RENAME FLGHT400.Customer_Names TO SYSTEM NAME CUSTNAMES;

CREATE OR REPLACE VIEW FLGHT400.Customer_Order (ORDER_NUMBER, AGENT_NO,
CUSTOMER_NO, FLIGHT_NUMBER, DEPARTURE_DATE, TICKETS_ORDERED, CLASS,
SEND_SIGNATURE_WITH_ORDER) AS
select * from customer_orders;

RENAME FLGHT400.Customer_Order TO SYSTEM NAME CUSTORDER;
```

By keeping the base tables unqualified, you can create different versions of the physical layer database for testing purposes. Run the script and verify that it ran successfully.

When the view is in place, all future access is performed against the view. There might be some exceptions, but these exceptions are part of the data access module and hidden from users and programmers. During the creation of the view, the location of the base table is resolved. When it accesses the view, DB2 for i finds the base table in the future schema (Flight_DB2). This means that the schema that contains the physical objects does not have to be included in any library lists.

Using a **SELECT *** against a view is an exception to the general rule of avoiding **SELECT ***. This general rule is true if the base table is specified on the FROM clause. This is meant to reduce database calls, reduce memory requirements, and encourage index only access use. All of these benefits apply to the **SELECT** contained within the view.

Another benefit to the programmer is the ability to soft code the columns that are contained within a view. This ability can become useful, especially if existing columns are modified within the view or columns are added to the view. The data access module that is associated with the view simply must be recompiled.

A single view can be used in several situations. Each situation might require different selection or ordering. These variations can be tuned externally by adding appropriate indexes, without the need to change or recompile existing programs.

A view does not contain data. Thus, you cannot create an index over a view. This prevents the view from being accessed randomly (a good thing). SQL does provide methods for returning a single row from a view. It is the DB2 for i optimizer's job to determine the best way to access the data.

The view name should be meaningful. The main purpose of the SQL View is for use in the bridge process. This means that the view might always need to contain all of the columns that were defined in the original file. For example, the original file might contain a numeric date column that is used to identify the date that the row was initially created. At some point, this numeric field becomes a date type. The view transforms the date type back to the original form to satisfy the original program requirement. However, newer views of the table are used for future development where the date field is used directly. The system name of the view must be used to soft code the fields (that is, externally described structures).

It is important to note that, other than managing the objects, an SQL view does not add any system processing impact. On IBM i, the SQL View is simply a container for an SQL statement. During execution, the SQL statement is merged with the SQL query to form a composite statement. This composite is then optimized to create the SQL execution plan.

9.3.5 Creating and deploying the SQL I/O modules

There are two types of SQL stored procedures: SQL PL or External. The SQL PL stored procedure is written entirely in SQL, and an External SQL procedure is written in a host language such as RPG IV.

There are several good reasons to use SQL PL to create the data access layer:

- ▶ Portability of code
- ▶ Usage of named arguments and global parameters
- ▶ Global variables
- ▶ Modularity by using functions and triggers
- ▶ Ease of use in coding dynamic SQL with descriptors

The data development project

The data perspective that is provided with IBM Data Studio, InfoSphere Data Architect, and some other Rational products include wizards and tools for quickly creating and deploying SQL routines.

In our example, we create a data development project in the workspace that contains the Flight400 database reengineering models. To do this, right-click within the Data Project Explorer and select **New** → **Data Development Project** to start the create Data Development Project tool. Enter "Phase 2 Data Access Layer" in the Project name field and then click **Next**, as shown in Figure 9-37 on page 397.

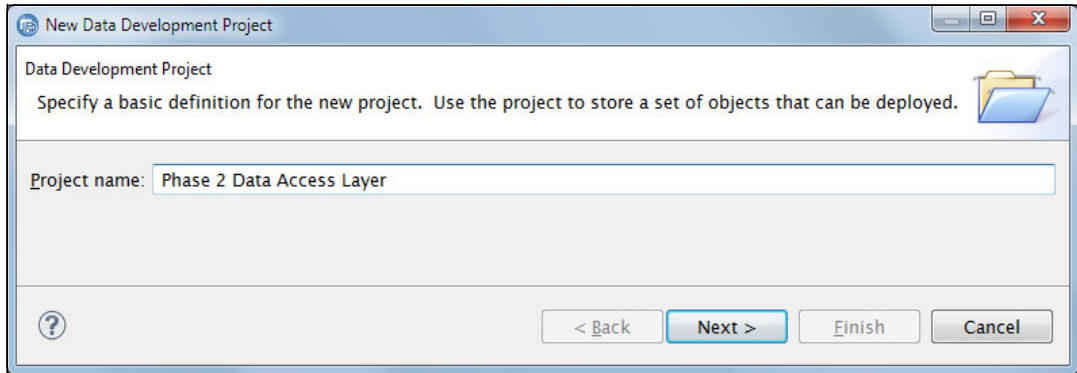


Figure 9-37 Creating a data development project

Select the database connection from the list of connections and click **Next**, as shown in Figure 9-38.

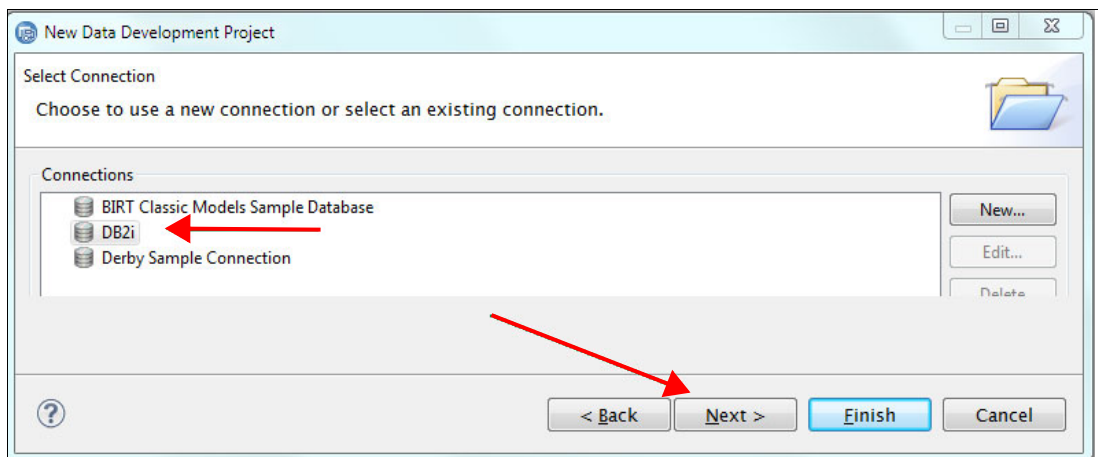


Figure 9-38 Selecting a database connection from the list of connections

Enter FLGHT400M as the name of the Default schema, which is the same as the schema containing the SQL views. Enter FLGHT400M as the schema name for the default path, as shown in Figure 9-39. DB2 for i searches the default schema for all database objects (tables, views, sequences, and so on). DB2 for i searches the path for SQL routines.

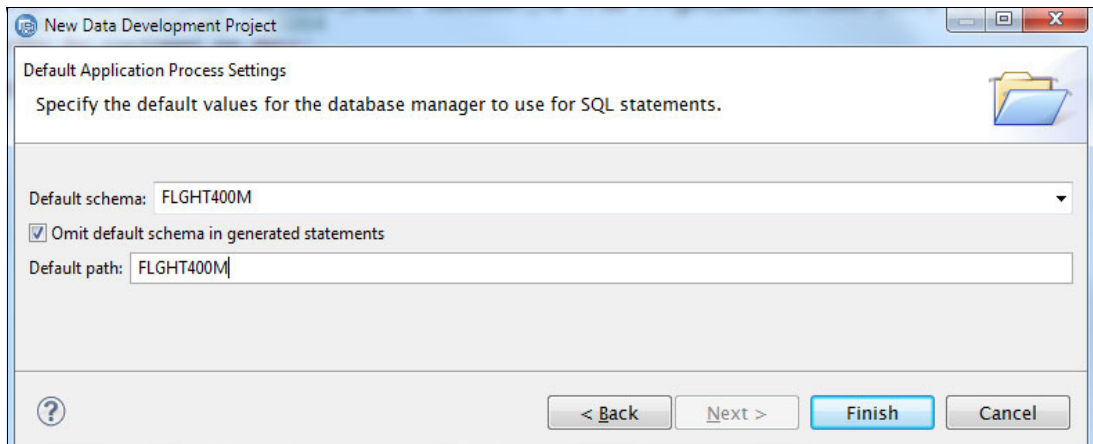


Figure 9-39 Specifying a name for the default schema name

The new data development project is now visible in the Data Project Explorer, as shown in Figure 9-40.

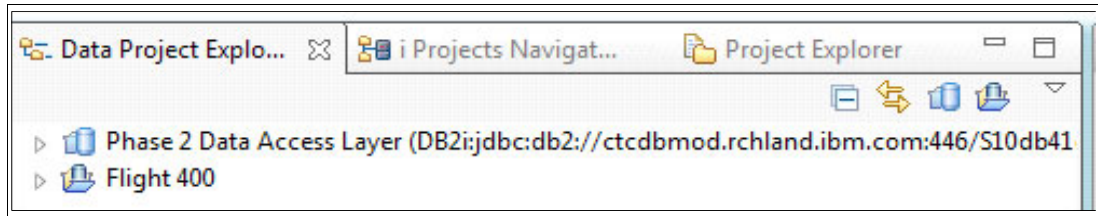


Figure 9-40 Phase 2 Data Access Layer within the Data Project Explorer view

Identity column versus sequence object

In our data model, we define our Primary key as an identity column. The Flight/400 application has defined CUSTOMER_NO as a UNIQUE key and provides application-centric methods to generate a customer number when a customer is added. As part of our Phase 2 isolation effort, we want to replace this type of application method with a more data-centric approach. You cannot define more than one identity column per table, so you must use a different SQL tool. We chose a sequence object.

Table 9-3 shows the differences between a sequence object and an identity column.

Table 9-3 Differences between a sequence object and an identity column

Identity column	Sequence object
An identity column can be defined as part of a table when the table is created or it can be added to a column by using an ALTER TABLE statement. After a table is created, the identity column characteristics can be changed.	A sequence object is a system object of type *DTAARA that is not tied to a table.
An identity column automatically generates values for a single table.	A sequence object generates sequential values that can be used in any SQL statement.
When an identity column is defined as GENERATED ALWAYS , the values that are used are always generated by the database manager. Applications are not allowed to provide their own values when changing the contents of the table. If the identity column is defined as BY DEFAULT , a value can be provided by the application.	There are two expressions that are used to retrieve the next values in the sequence and to look at the previous value that is assigned for the sequence. The PREVIOUS VALUE expression returns the most recently generated value for the specified sequence for a previous statement within the current session. The NEXT VALUE expression returns the next value for the specified sequence. The usage of these expressions allows the same value to be used across several SQL statements within several tables.
The IDENTITY_VAL_LOCAL function can be used to see the most recently assigned value for an identity column. In addition, the generated value can be returned as a column in the projected result set of the FINAL TABLE when INSERT on FROM is used.	

Creating SQL scripts

To determine the starting value for the customer number sequence for our example, we had to determine the last customer number that was used. To do this, right-click **SQL Scripts** from within the Phase 2 Data Access Layer project and select **New** → **SQL or XQuery Script** to start the wizard. Enter `Query_Customer_Names` in to the Name field and then click **Finish** to open the SQL script editor, as shown in Figure 9-41.

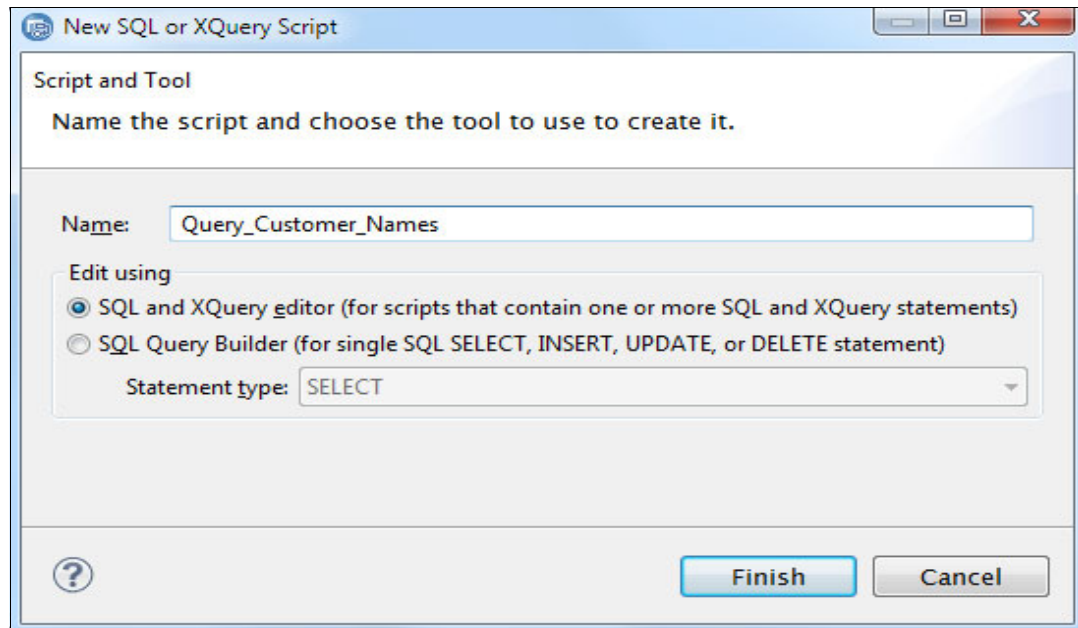


Figure 9-41 Specifying the SQL or XQuery name

Enter the following SQL statements to determine that the last customer number is used. By default, the current schema defaults to the user profile name. To get around this, use **SET CURRENT SCHEMA** to change the default schema to our schema, `FLGHT400M`. This is shown in Example 9-14.

Example 9-14 Changing the default schema

```
SET CURRENT SCHEMA FLGHT400M;  
  
SELECT MAX(CUSTOMER_NO) AS LAST_CUSTOMER_NO  
FROM CUSTOMER_NAMES;
```

Review the SQL Results and record the value that was displayed (15004 for `LAST_CUSTOMER_NO`) in the example. Close the SQL Script window.

Create another SQL script (for example, `CreateSequence_Customer_Number`), which constrains the SQL **DDL** statement to create a sequence object. This object is used to contain the next value for `CUSTOMER_NO`. Example 9-15 shows the SQL **DDL** that we used.

Example 9-15 Script to constrain the SQL **DDL** statement to create a **SEQUENCE** object

```
CREATE OR REPLACE SEQUENCE FLGHT400M.CUSTOMER_NUMBER  
START WITH 15005  
INCREMENT BY 1  
NO MAXVALUE  
NO CYCLE  
CACHE 24;
```

We are starting with 15005, which is one more than the last customer number that was created. Run the script and review the SQL Results to ensure that the sequence object is created. You can now create the first data access module.

SQL stored procedures

In our example, you can work with the SQL stored procedures by right-clicking **Stored Procedures** from within the Phase 2 Data Access Layer project and selecting **New** → **Stored Procedure** to start the wizard. Enter `Add_New_Customer` in the Name field, select **Custom** from the list of templates, and click **Finish** to open the editor, as shown in Figure 9-42.

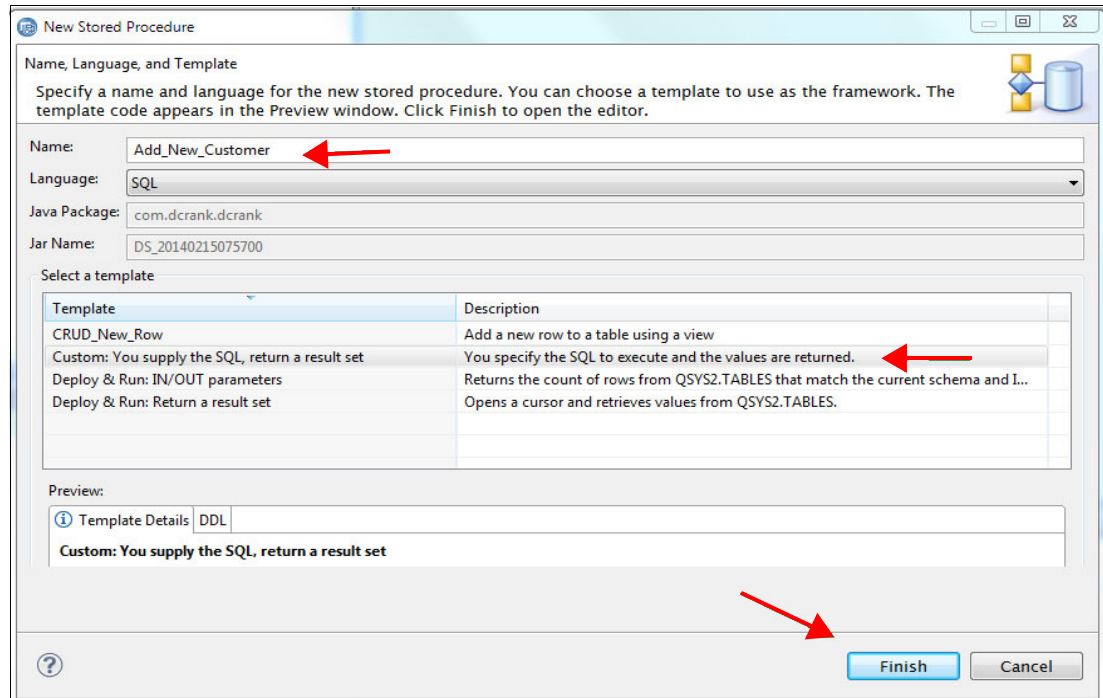


Figure 9-42 New Stored Procedure wizard

The templates that are provided with the Data Perspective can be customized. In addition, you can add your own templates as needed. To do this task, expand and open the **Preferences** view from the window menu bar. Find and expand **SQL Development** → **Routines** → **Templates**. Find the DB2 for i SQL templates by reviewing the Context column and finding any item with the context `db2i5_sp_sql`, as shown in Figure 9-43.

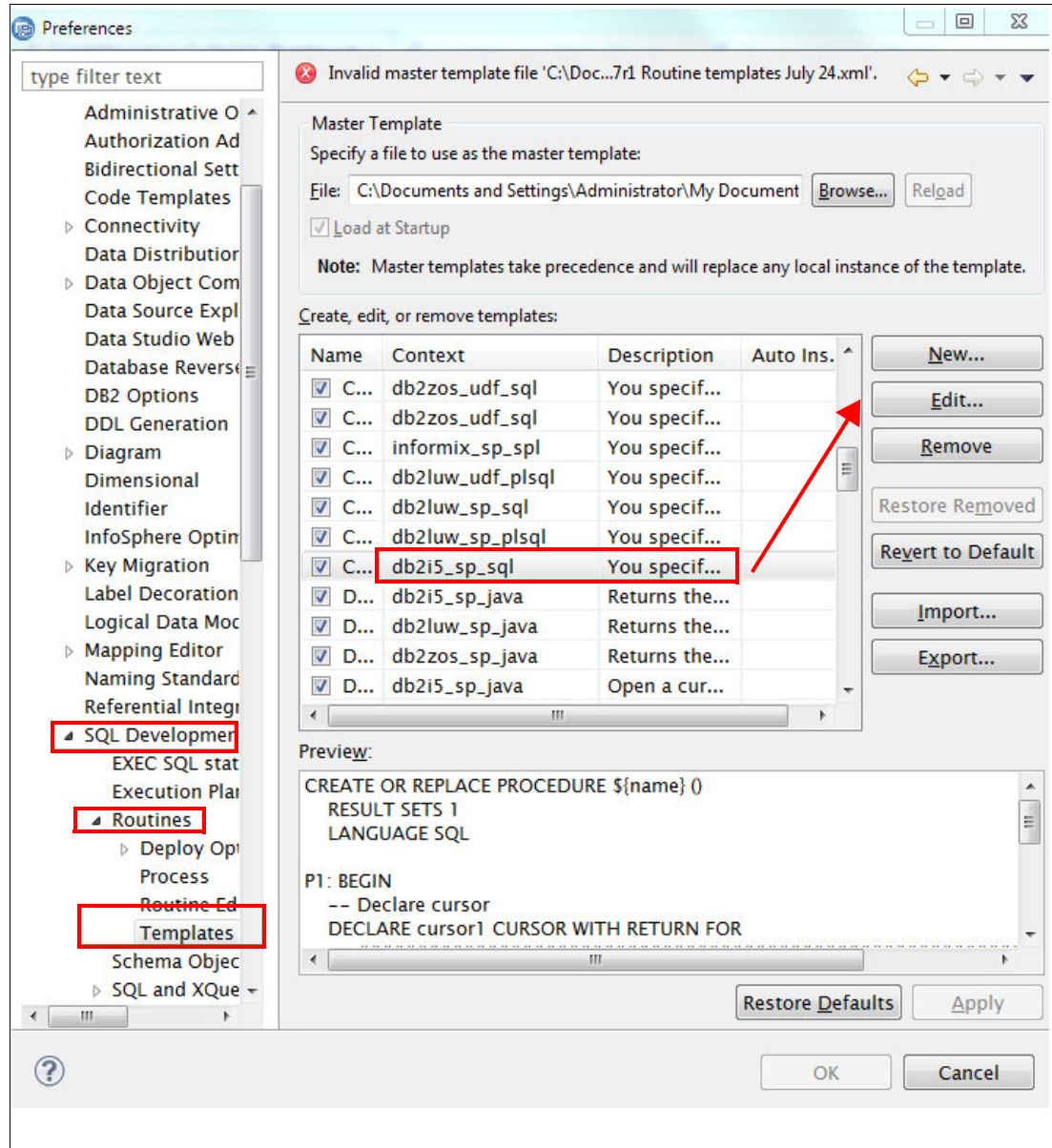


Figure 9-43 Updating the `db2i5_sp_sql` context

Highlight the item that you want to change and click **Edit** to open the Edit Template window. Change CREATE to CREATE OR REPLACE, as shown in Figure 9-44.

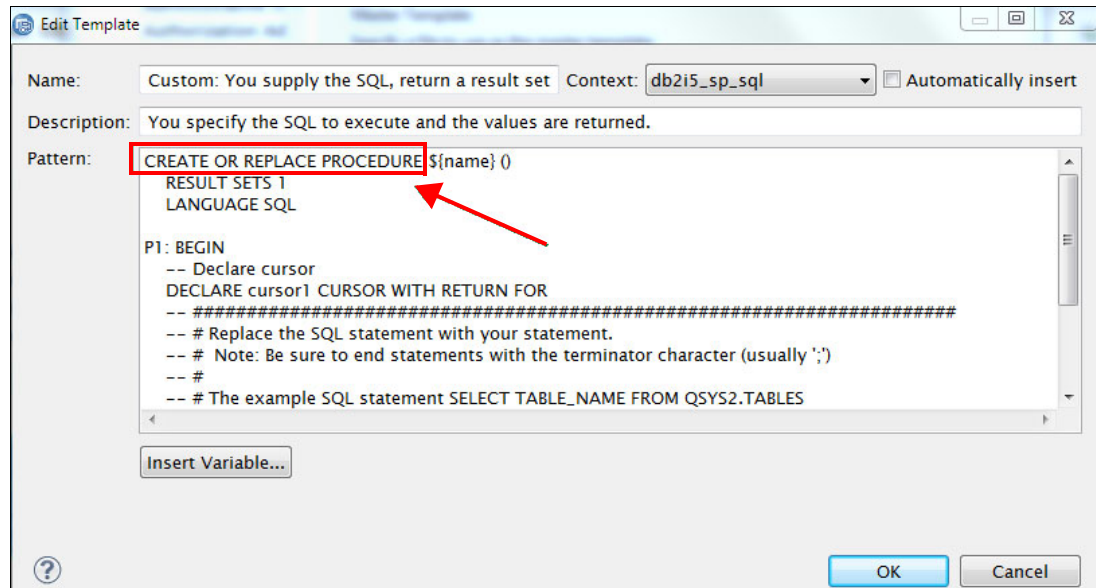


Figure 9-44 Updating the SQL

In our example, we replace the generated body of the SQL procedure with the SQL that is shown in Example 9-16.

Example 9-16 Updated SQL for the body of the request

```

CREATE OR REPLACE PROCEDURE Add_New_Customer (
  p_Customer_Name VARCHAR(64),
  p_Customer_No INTEGER DEFAULT(NEXT VALUE FOR Customer_Number))

LANGUAGE SQL
SPECIFIC ADDNEWCUST
COMMIT ON RETURN YES
PROGRAM TYPE SUB

P1: BEGIN

  INSERT INTO customer_names (CUSTOMER_NO, CUSTOMER_NAME)
    VALUES(p_Customer_No,p_Customer_Name);
END P1

```

We create our procedure as a service program (PROGRAM TYPE SUB). Commitment control is normally the default for external connections and program compiler settings, thus ensuring that the data is committed by adding the **COMMIT ON RETURN YES** option.

Define one input parameter for each column in the Customer_Names view, beginning with the required parameters first, followed by the optional parameters. Define DEFAULT values for the optional parameters. The default for p_Customer_No uses the **NEXT VALUE FOR** function to retrieve the next customer number.

Although not required, you can code the column names from the view as part of the **INSERT INTO** statement. Do this for readability and future maintenance.

Deploying and testing

Deploy the procedure by right-clicking **ADD_NEW_CUSTOMER** from the Stored Procedures list and selecting **Deploy...** Verify that the Target schema and Default path are both FLGHT400M and click **Finish** to deploy the stored procedure. Verify that the SQL Stored Procedure is created successfully.

In our example, we create a SQL script that is named TestStoredProcedure_ADD_NEW_CUSTOMER. The script contained the SQL statements found in Example 9-17.

Example 9-17 TestStoredProcedure_ADD_NEW_CUSTOMER SQL script

```
SET SCHEMA FLGHT400M;
SET PATH FLGHT400M;

CALL ADD_NEW_CUSTOMER ('Traveling Man');

SELECT CUSTOMER_NO, CUSTOMER_NAME, CUSTOMER_ID,
       ROW CHANGE TIMESTAMP FOR T1 AS When_Added
  FROM FLIGHT_DB2.CUSTOMER_INFO AS T1
 WHERE CUSTOMER_NO > 15004;
```

The **CALL** statement contains only the required parameters. The unique key value for the new row is automatically generated.

Figure 9-45 shows the results of running the test script.

Status	Result1			
	CUSTOMER_NO	CUSTOMER_NAME	CUSTOMER_ID	WHEN_ADDED
1	15005	Traveling Man	20053	2014-02-15 09:55:19.508773

Figure 9-45 Results of the SQL call

Read-only procedures

The data structures that are contained within the Flight/400 database are well-designed for a relational database; however, except for a couple of DDS join logical files, most of the data access was using “record at a time” methods (that is, READ and CHAIN in RPG). Our new data access layer uses SQL set based methods, such as joins to access data from more than one table.

Using SQL joins

An SQL join is the method that is used to combine, intersect, or differentiate sets of data. It is based on the branch of mathematics that is known as “set theory.” The good news is that DB2 for i does the math; you code SQL joins and let DB2 for i do the heavy lifting.

DB2 for i supports the join types that are described in Table 9-4.

Table 9-4 DB2 for i join types and details

Join type	HLL equivalent	Usage
INNER	READ + RANDOM READ (CHAIN in RPG)	Continue if found. Matching rows only.
OUTER (right or left)	READ + RANDOM READ (CHAIN in RPG)	Continue if not found, Match and unmatched rows. Provide default value for unmatched columns (SQL is NULL).
EXCEPTION (right or left)	READ + RANDOM READ (CHAIN in RPG)	Continue if not found. Unmatched rows only.
CROSS	READ + READ	For each row read from primary file, read all rows for secondary file, matched and unmatched.
FULL	READ + RANDOM READ + READ	For primary file, read all matched and unmatched rows. For secondary file, matched and unmatched rows

As our new data model becomes more normalized, our SQL joins become more complex. We choose to encapsulate our joins in views because this simplifies the coding of both our SQL and External Stored Procedures.

Our first SQL joined view is based on the DDS join logical file CUSTORD. Table 9-5 compares the DDS and DDL version of the join.

Table 9-5 DDS versus DDL join

DDS	DDL
R CUSNMR JFILE(CUSTOMERS ORDERS) J JFLD(CUST000001 CUST000001) ORDRNO RENAME(ORDER00001) AGNTNO RENAME(AGENT_NO) CUSTNM RENAME(CUST000002) CUSTNO RENAME(CUST000001) JREF(CUSTOMERS) FLGTNO RENAME(FLIGH00001) DEPART RENAME(DEPAR00001) TCKTNO RENAME(TICKE00001) CLASS SENDSG RENAME(SEND_00001) K CUSTNM K CUSTNO	SELECT ORDER_NUMBER, AGENT_NO, CUSTOMER_NAME, T1.CUSTOMER_NO, FLIGHT_NUMBER, DEPARTURE_DATE, TICKETS_ORDERED, CLASS, SEND_SIGNATURE_WITH_ORDER FROM CUSTOMER_INFO T1 JOIN CUSTOMER_ORDERS T2 ON T1.CUSTOMER_NO = T2.CUSTOMER_NO;

There are a number of differences between the syntax that is shown in Table 9-5 of the source code, which are listed here:

- ▶ The SQL join is coded at the end of source versus the beginning of the DDS.
- ▶ The DDS short names were required to support older versions or RPG. Going forward, use the long names that are defined in the DDS.
- ▶ The T1 correlation name for CUSTOMER_INFO is equivalent to the JREF DDS keyword.
- ▶ The key information is not part of the view. The purpose of the keys is not apparent in the DDS (that is, is it used for selection, ordering, or both).

To create the join views, use the Phase 2 Customer Orders with Views diagram to define the views and to generate the DDL. Figure 9-46 is the new version of the diagram after adding the joined view.

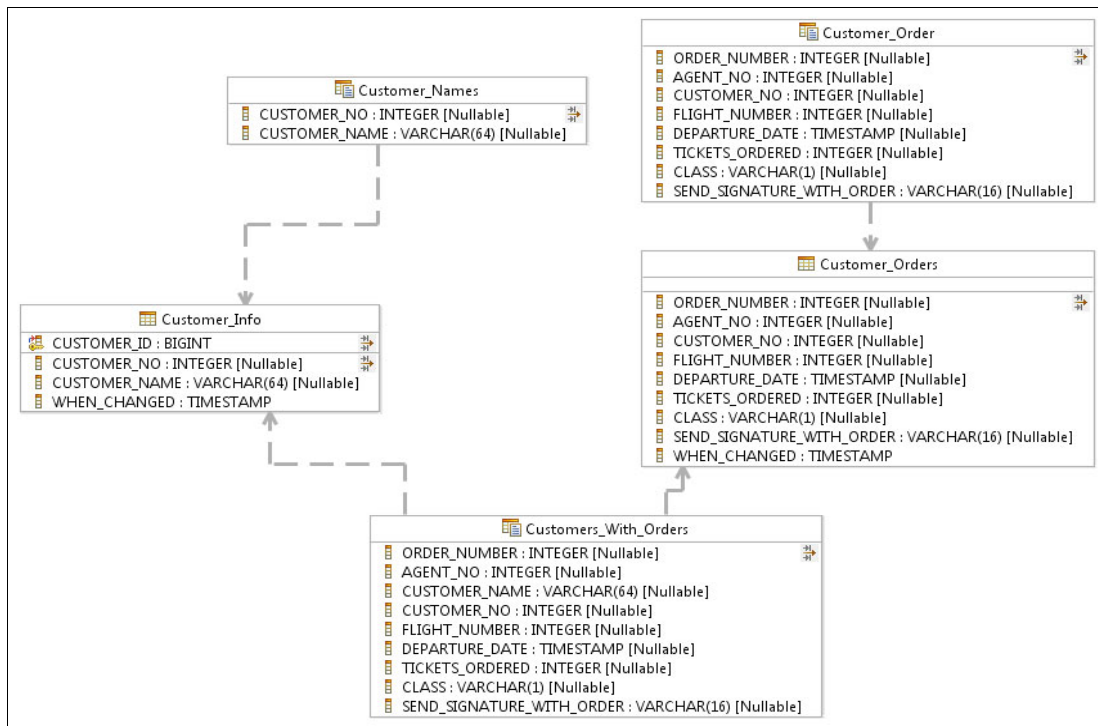


Figure 9-46 New Phase 2 Diagram after adding the joined view

Example 9-18 shows the generated DDL (with our modifications) for the Customer_With_Orders view.

Example 9-18 Generated DDL with modifications

```

SET SCHEMA FLIGHT_DB2;

CREATE OR REPLACE VIEW FLGHT400M.CUSTOMERS_WITH_ORDERS
(ORDER_NUMBER, AGENT_NO, CUSTOMER_NAME, CUSTOMER_NO, FLIGHT_NUMBER,
DEPARTURE_DATE,
TICKETS_ORDERED, CLASS, SEND_SIGNATURE_WITH_ORDER) AS
SELECT ORDER_NUMBER, AGENT_NO, CUSTOMER_NAME, T1.CUSTOMER_NO, FLIGHT_NUMBER,
DEPARTURE_DATE,
TICKETS_ORDERED, CLASS, SEND_SIGNATURE_WITH_ORDER
FROM CUSTOMER_INFO T1 JOIN CUSTOMER_ORDERS T2 ON T1.CUSTOMER_NO = T2.CUSTOMER_NO;

RENAME FLGHT400M.CUSTOMERS_WITH_ORDERS TO SYSTEM NAME CUSTWORDRS;

```

To test the view, return to the Data Source Explorer, expand the schema **FLGHT400M** → **Views**, and find **CUSTOMERS_WITH_ORDERS**, as shown in Figure 9-47.

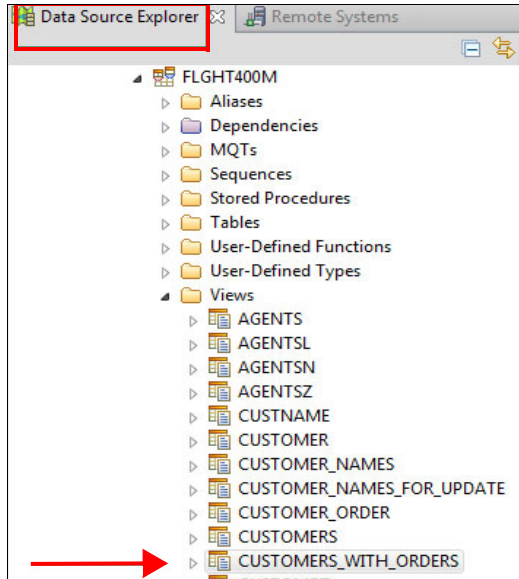


Figure 9-47 Data Source Explorer showing the **CUSTOMERS_WITH_ORDERS** selected

Right-click the view, select **Data** → **Sample Contents**, and review the results, as shown in Figure 9-48.

Status	Result1	ORDER_NUMBER	AGENT_NO	CUSTOMER_NAME	CUSTOMER_NO	FLIGHT_NUMBER	DEPARTURE_DATE	TICKETS_ORDERED	CLASS	SEND_SIGNAT
1		4971187	9	doe, john	10013	6112797	2004-06-12 07:40...	1	3	N
2		4981957	8	doe, john	10013	7127359	2004-05-09 07:40...	1	3	N
3		4987057	4444	doe, john	10013	1147232	2004-09-20 07:12...	1	3	N
4		5008985	9	doe, john	10013	1129762	2004-02-09 07:12...	1	3	N
5		5030572	2	doe, john	10013	2176114	2004-11-16 07:19...	1	3	N
6		5053381	4	doe, john	10013	7119440	2004-05-23 08:01...	1	3	N
7		5058084	5	doe, john	10013	7142643	2004-08-15 07:19...	1	3	N
8		5085454	7	doe, john	10013	7179857	2004-08-01 08:01...	1	3	N
9		5086511	5	doe, john	10013	6108217	2004-02-07 07:19...	1	3	N
10		5089061	7	doe, john	10013	1154748	2004-10-04 07:19...	1	3	N

Figure 9-48 Results of the Sample Contents view

After successfully testing the SQL view, you can create an SQL stored procedure to access the view.

Defining an SQL procedure to return a result set

We create our SQL stored procedure by using the same process that we used to create the **ADD_NEW_CUSTOMER** procedure. Name the new procedure **LIST_CUSTOMERS_WITH_ORDERS**. Define a one-character non-required parameter that is named **p_StartsWith** and define a default value of "A".

For options, allow the procedure to return one result set. In addition, provide a 10-character specific name and create the procedure as a service program. This process is shown here:

```
CREATE OR REPLACE PROCEDURE List_Customers_With_Orders (
  p_StartsWith CHAR(1) DEFAULT 'A' )
  RESULT SETS 1
  LANGUAGE SQL
  SPECIFIC LISTCUSORD
  PROGRAM TYPE SUB
```

```

P1: BEGIN
  -- Declare cursor
  DECLARE LISTCUSORD_cursor1 CURSOR WITH RETURN FOR

  SELECT * FROM Customers_With_Orders
  WHERE SUBSTR(Customer_Name,1) >= p_StartsWith
  ORDER BY Customer_Name, Customer_No;

  -- Cursor left open for client application
  OPEN LISTCUSORD_cursor1;
END P1

```

Use the specific name that is concatenated with cursor1 to provide a unique name for the cursor. This allows you to add quickly more result sets as needed. We initially choose the default **WITH RETURN** to make the result set available to the calling application. This allows the result set to be returned to either a host-centric or external application. You can use **RETURN TO CLIENT** on nested procedures to return the result set to the application that called the outermost procedure in the call stack.

Deploy the SQL procedure and it is ready for testing.

Testing the read-only SQL procedure

To test the stored procedure, return to the Data Source Explorer, expand **FLGHT400M** → **Stored Procedures**, and find **LIST_CUSTOMERS_WITH_ORDERS**. Right-click the procedure and select **Run**. Enter **B** as the value for the **P_STARTS_WITH** parameter and click **Run**, as shown in Figure 9-49.

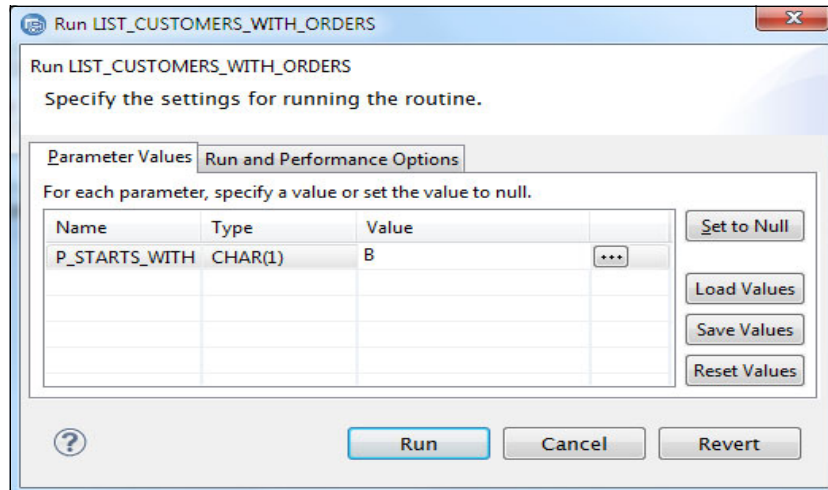


Figure 9-49 Run the LIST_CUSTOMERS_WITH_ORDERS test

The results for clicking **Run** are shown in Figure 9-50.

Status	Result1						
	ORDER_NUMBER	AGENT_NO	CUSTOMER_NAME	CUSTOMER_NO	FLIGHT_NUMBER	DEPARTURE_DATE	TICKETS_OR
1	4971187	9	doe, john	10013	6112797	2004-06-12 07:40...	1
2	4981957	8	doe, john	10013	7127359	2004-05-09 07:40...	1
3	4987057	4444	doe, john	10013	1147232	2004-09-20 07:12...	1
4	5008985	9	doe, john	10013	1129762	2004-02-09 07:12...	1
5	5030572	2	doe, john	10013	2176114	2004-11-16 07:19...	1
6	5053381	4	doe, john	10013	7119440	2004-05-23 08:01...	1
7	5058084	5	doe, john	10013	7142643	2004-08-15 07:19...	1
8	5085454	7	doe, john	10013	7179857	2004-08-01 08:01...	1
9	5086511	5	doe, john	10013	6108217	2004-02-07 07:19...	1
10	5089061	7	doe, john	10013	1154748	2004-10-04 07:19...	1

Figure 9-50 Results of running the SQL for LIST_CUSTOMERS_WITH_ORDERS

Repeat the test with various starting characters. In all cases, you must scroll through the result set to verify that the data was ordered correctly. You cannot test the default value of A using this method because the stored procedure passes a blank; however, we discover that our data was not consistent. Figure 9-51 shows the result set when you do not provide a value for p_StartsWith.

Status	Parameters	Result1					
	ORDER_NUMBER	AGENT_NO	CUSTOMER_NAME	CUSTOMER_NO	FLIGHT_NUMBER	DEPARTURE_DATE	TICKETS_ORDERED
1	4971187	9	doe, john	10013	6112797	2004-06-12 07:40...	1
2	4981957	8	doe, john	10013	7127359	2004-05-09 07:40...	1
3	4987057	4444	doe, john	10013	1147232	2004-09-20 07:12...	1
4	5008985	9	doe, john	10013	1129762	2004-02-09 07:12...	1
5	5030572	2	doe, john	10013	2176114	2004-11-16 07:19...	1
6	5053381	4	doe, john	10013	7119440	2004-05-23 08:01...	1
7	5058084	5	doe, john	10013	7142643	2004-08-15 07:19...	1
8	5085454	7	doe, john	10013	7179857	2004-08-01 08:01...	1

Figure 9-51 Results when no value is specified for p_STARTS_WITH

Based on these results, we need to make some updates.

Updating and deleting a procedure

When creating the update and delete procedures, we took advantage of data-centric programming techniques, which included Optimistic Locking and Global Variable support.

Optimistic locking

One of the reasons that we add the ROW CHANGE TIMESTAMP column is to take advantage of the concept of optimistic locking. The idea is that multiple rows can be accessed for read-only; however, at the same time, we want to include the data that is necessary to perform a searched update. Lock only rows when an update is requested.

It is possible that a row read for input might be updated by another process. If the current process updates the same row, the prior update might be lost. To avoid this, capture the ROW CHANGE TIMESTAMP value at read and then use that value as an additional argument, along with the UNIQUE key, on the WHERE clause. If the row is not found, it has either been updated by another process or deleted.

Here is an example of the technique:

Read some rows, which include the unique key (Customer_No) and the row change time stamp. Because there is only one row change time stamp per table, you can use the **ROW CHANGE TIMESTAMP** function without knowing the column name, as shown here:

```
SELECT customer_no, customer_name,  
       ROW CHANGE TIMESTAMP FOR T1 AS v_When_Changed  
   INTO v_customer_no, v_customer_name, v  
 FROM Flight_db2.CUSTINFO T1  
WHERE customer_name LIKE 'A%';
```

At some point, we decide to update one of the returned rows. The WHERE clause contains the host variable for the unique key and the row change time stamp, as shown here:

```
UPDATE Flight_db2.CUSTINFO U1  
SET customer_name = 'Frequent Flyer'  
WHERE customer_no = v_Customer_No  
   AND ROW CHANGE TIMESTAMP FOR U1 = v_When_Changed;
```

Global variables

We take advantage of another modern SQL technique that is known as *global variables*. A global variable is an object that is defined by using the **CREATE VARIABLE SQL DDL** statement. The definition is stored permanently but contains no data, similar to an SQL view. When the global variable is referenced within a session, a temporary object is created for that session only. The global variable can be defined with default values and can be modified by using standard SQL statements.

Here is a link to an article that contains more information about global variables:

<http://iprodeveloper.com/database/think-globally-run-locally-db2-i-71>

Create an SQL script named CreateGlobalVariable_gv_When_Changed. This global variable is populated by the application and referenced by the SQL view that is used for update and delete operations. We decide to treat our global variables as application objects and thus create the global variable in our virtual layer schema.

Here is the code that we use to define the global variable:

```
CREATE OR REPLACE VARIABLE FLGHT400M.GV_WHEN_CHANGED TIMESTAMP DEFAULT NULL;  
  
COMMIT;
```

The Read-only and Update SQL view

We update our Phase 2 Customer Orders with Views by adding two new SQL views that are named Customer_Names_With_Last_Changed_Info and Customer_Names_For_Update. Besides the non-hidden columns, we also need the row change time stamp column, as part of the projected result set for read and as an additional argument for update.

The SQL statement for the read-only view was coded as follows:

```
SELECT T1.CUSTOMER_NO, T1.CUSTOMER_NAME,  
       ROW CHANGE TIMESTAMP FOR T2 AS Last_Changed  
 FROM Customer_Info T1  
JOIN Customer_Info T2 ON t1.Customer_No = T2.Customer_No;
```

Use a self-join to retrieve the row change time stamp for Customer_info to make the view non-updateable. Change the name of the row change time stamp column to keep the original hidden.

The SQL statement for the update view was coded as follows:

```
SELECT Customer_no, Customer_name, ROW CHANGE TIMESTAMP FOR CUSTINFO AS
WHEN_CHANGED FROM Customer_Info CUSTINFO.
```

Make the row change time stamp column part of the projected result set so that you can reference it on a WHERE clause.

The diagram now looks like Figure 9-52.

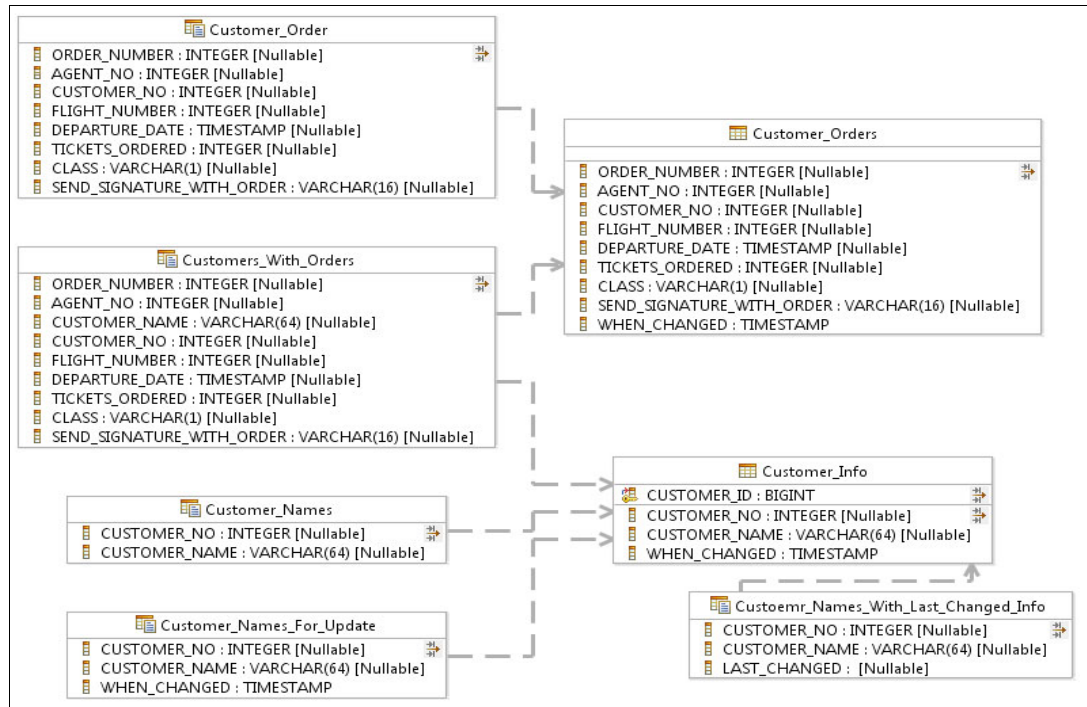


Figure 9-52 Phase 2 updated SQL diagram

Generate the DDL for the new views and open each view for editing. Modify both SQL script scripts by adding the following items:

- ▶ A **SET SCHEMA** statement for our physical database layer
- ▶ An **OR REPLACE** clause between the **CREATE** and **VIEW**

The script for the Customer_Names_With_Last_Changed_Info is shown below:

```
SET SCHEMA Flight_DB2;
```

```
CREATE OR REPLACE VIEW FLGHT400M.Customer_Names_With_Last_Changed_Info
(CUSTOMER_NO, CUSTOMER_NAME, LAST_CHANGED) AS
SELECT T1.CUSTOMER_NO, T1.CUSTOMER_NAME,
      ROW CHANGE TIMESTAMP FOR T2 AS Last_Changed
FROM Customer_Info T1
JOIN Customer_Info T2 ON t1.Customer_No = T2.Customer_No;
```

```
RENAME FLGHT400M.Customer_Names_With_Last_Changed_Info TO SYSTEM NAME
CUSTWRCTS;
```

- ▶ A **SET PATH** statement specifying our virtual layer schema to access the global variable
- ▶ A **WHERE** clause referencing our row change time stamp global variable that was created previously

The script for Customer_Names_For_Update is shown below:

```
SET SCHEMA Flight_DB2;
SET PATH FLGHT400M;

CREATE OR REPLACE VIEW FLGHT400M.Customer_Names_For_Update
(CUSTOMER_NO, CUSTOMER_NAME, WHEN_CHANGED) AS
SELECT Customer_no, Customer_name,
       ROW CHANGE TIMESTAMP FOR CUSTINFO AS WHEN_CHANGED
FROM Customer_Info CUSTINFO
WHERE WHEN_CHANGED = gv_When_Changed;

RENAME FLGHT400M.Customer_Names_For_Update TO SYSTEM NAME CUSTNAMESU;
```

The SQL Update and Delete procedures

Before creating our Update and Delete procedures, we realize there are create options that are common to all procedures. We take advantage of the Snippet tool to create a template that can be reused and improve productivity.

Reusing code through Snippets

To use the Snippets tool, you must add the Snippets view to the workspace by clicking **Window** → **Show View** → **Other** and then expanding **General**. Click **Snippets** to add the view to the bottom window in the workspace. Right-click the first item in the Snippets view and select **Customize**. The **Customize Palette** window opens, as shown in Figure 9-53.

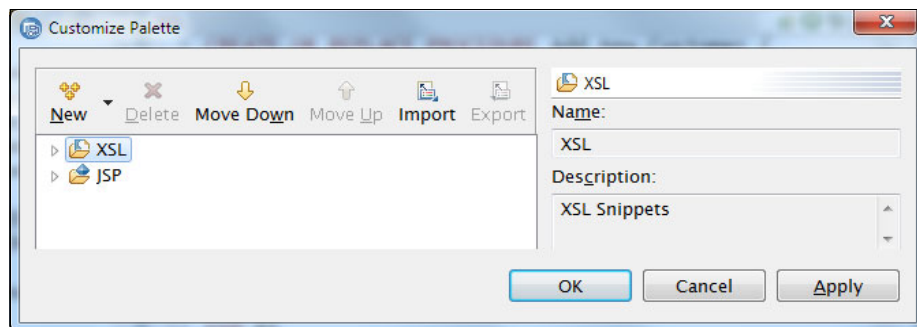


Figure 9-53 Customize Palette window for maintaining Snippets

Click **New** → **New Category** to add a new Unnamed Category to the list. Change the name from Unnamed Category to My SQL Snippets, click **Apply**, and click **OK**. Open the Add_New_Customer procedure and copy the options starting at SPECIFIC through PROGRAM TYPE SUB.

Return to the Snippets view, right-click **My SQL Snippets**, and select **Paste As Snippet**. This selection added an Unnamed Template to the template list under My SQL Snippets and opened the template for editing.

Change the name from Unnamed Template to OPTION List for Non-Select. Add the following description: “Common options for INSERT/UPDATE/DELETE procedures”. The Customize palette window now looks like Figure 9-54.

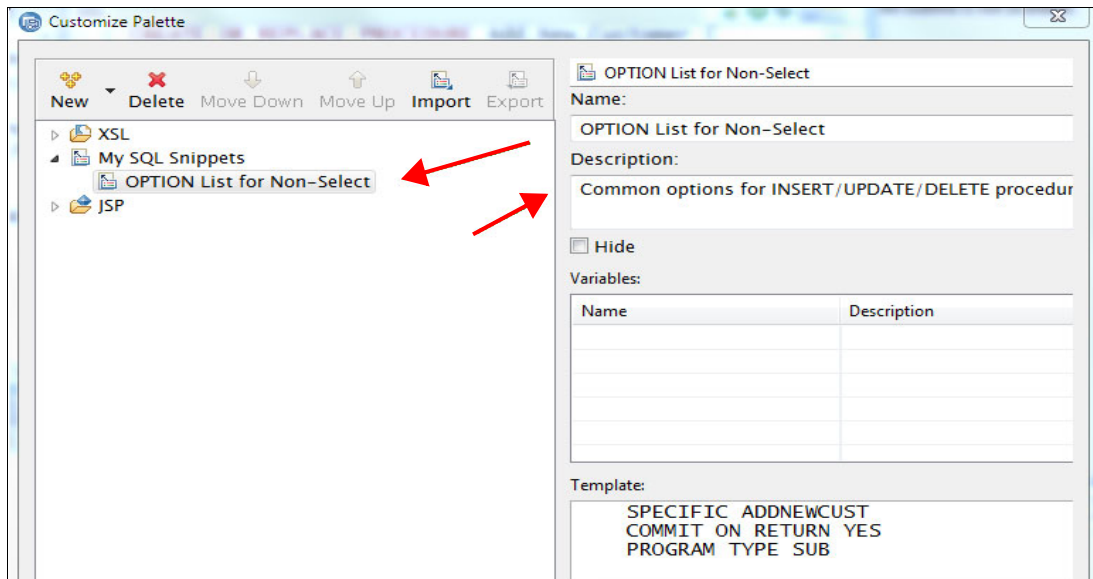


Figure 9-54 Updated Customize Palette window

One of the more powerful capabilities of Snippets is its ability to define variables that can be modified when the snippet is added to a source member. Use this feature to make the SPECIFIC name a variable. To do this task, click the **New** button that is to the right of the Variable list pane. Enter SpecificName as the variable name and “System name for procedure” as the Description. In the Template code, delete ADDNEWCUST and in its place click **Insert Variable Placeholder** and select **SpecificName** as the variable. Click **Apply** and then **OK** to complete the snippet, as shown in Figure 9-55.

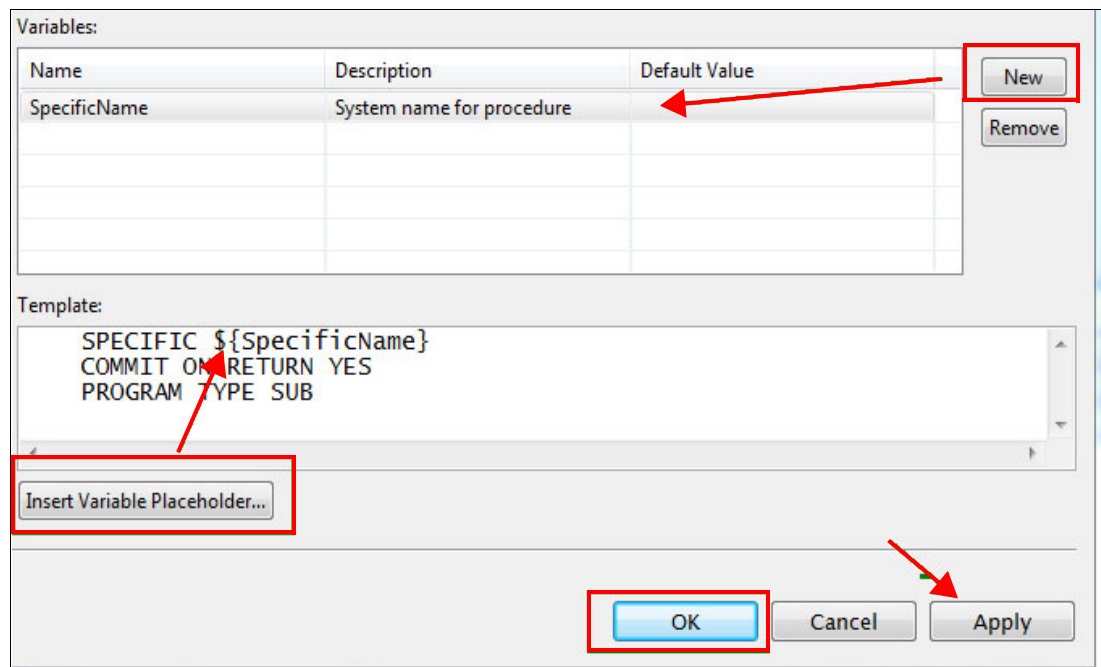


Figure 9-55 Updating the snippet code

Using the SQL Procedure wizard, create a procedure that is named `Update_Customer_Names`. Define the unique key (`Customer_No`), each updatable column (`iCustomer_Name` in our case) as input parameters, and a parameter containing the retrieved row change time stamp (`p_When_Changed`). Remove the auto-generated `RESULT SETS 1` option and insert the new snippet in its place by right-clicking the snippet and selecting **Insert**. When prompted for the `SpecificName`, enter `UPDCUSTNME`.

In the body of the stored procedure, code a **SET** statement to update the global variable **`gv_When_Changed`** with the parameter **`p_When_Changed`**. Follow this with an **UPDATE** statement to modify the customer name based on the value in the **`p_Customer_Number`** and **`p_Customer_Name`** parameter values:

```
CREATE OR REPLACE PROCEDURE Update_Customer_Names (  
    p_Customer_No INTEGER,  
    p_Customer_Name VARCHAR(64),  
    p_When_Changed TIMESTAMP)  
LANGUAGE SQL  
SPECIFIC UPDCUSTNME  
COMMIT ON RETURN YES  
PROGRAM TYPE SUB
```

```
P1: BEGIN
```

```
    SET FLGHT400M.gv_When_Changed = p_When_Changed;
```

```
    UPDATE Customer_Names_For_Update  
        SET customer_name = p_Customer_Name  
    WHERE customer_no = p_Customer_No;
```

```
END P1
```

Create another SQL stored procedure that is named `Delete_Customer` to handle customer deletes. It is similar to the update procedure without the **`p_Customer_Name`** parameter. It is shown here:

```
CREATE OR REPLACE PROCEDURE Delete_Customer (  
    p_Customer_No INTEGER,  
    p_When_Changed TIMESTAMP)  
LANGUAGE SQL  
SPECIFIC DLTCUSTOMR  
COMMIT ON RETURN YES  
PROGRAM TYPE SUB
```

```
P1: BEGIN
```

```
    SET gv_When_Changed = p_When_Changed;
```

```
    DELETE Customer_Names_For_Update  
    WHERE customer_no = p_Customer_No;
```

```
END P1
```

Testing the SQL Read-Only, Update, and Delete procedures

In our example, we decide to create an SQL stored procedure to simulate host-centric and external applications that use the new SQ I/O modules. Again, use the SQL Stored Procedure wizard to create a procedure that is named `Get_And_Update_Customer` with a specific name of `GETCUSTOMR`. Define the customer number as a required parameter and customer name as optional. Add another optional parameter to indicate the type of operation (that is, R for Read-only, U for Update and D for Delete) with read-only being the default. In addition, allow the procedure to return two result sets.

Here is the procedure:

```
CREATE OR REPLACE PROCEDURE Get_And_Update_Customer (  
    p_Customer_Number INTEGER,  
    p_Customer_Name VARCHAR(64) DEFAULT NULL,  
    p_Request_Type CHAR(1) DEFAULT 'R')  
    RESULT SETS 2  
    LANGUAGE SQL  
    SPECIFIC GETCUSTOMR  
    PROGRAM TYPE SUB
```

In the body of the procedure, define a variable to store the value of the current row change time stamp. Also, define two cursors, one for the read-only operation and a second to return the result of the update or delete operation. Here are those changes:

```
P1: BEGIN  
    -- Declare variables  
    DECLARE v_Last_Changed TIMESTAMP;  
    -- Declare cursor  
    DECLARE GETCUSTOMR_cursor1 CURSOR WITH RETURN FOR  
        SELECT * FROM Customer_Names  
            WHERE Customer_No = p_Customer_Number;  
    DECLARE GETCUSTOMR_cursor2 CURSOR WITH RETURN FOR  
        SELECT *  
            FROM Customer_Names_With_Last_Changed_Info  
            WHERE Customer_No = p_Customer_Number;
```

Use a **CASE** statement to control the execution of the procedure code depending on the type of operation. If the operation is read-only, then it returns the result set for cursor1. If the operation is an update or delete, then it opens cursor1 to return the before image, capture the current value for the row change time stamp, call the update or delete procedure, and then return the result set for cursor2. Here is the procedure:

```
    -- Perform appropriate action  
    CASE p_Request_Type  
        WHEN 'R'  
            THEN OPEN GETCUSTOMR_cursor1;  
        WHEN 'U'  
            THEN OPEN GETCUSTOMR_cursor1;  
            Select Last_Changed INTO v_Last_Changed  
                FROM Customer_Names_With_Last_Changed_Info  
                WHERE customer_no = p_Customer_Number;  
            Call UPDATE_CUSTOMER_NAMES (  
                p_Customer_Number,p_Customer_Name,v_Last_Changed);  
            OPEN GETCUSTOMR_cursor2;  
        WHEN 'D'  
            THEN OPEN GETCUSTOMR_cursor1;  
            Select Last_Changed INTO v_Last_Changed
```

```

FROM Customer_Names_With_Last_Changed_Info
      WHERE customer_no = p_Customer_Number;
Call DELETE_CUSTOMER (p_Customer_Number, v_Last_Changed) ;
OPEN GETCUSTOMR_cursor2;
END CASE;
END P1

```

Create a test script that is named TestStoredProcedure_Change_Customer within the Phase 2 data development project; however, we choose to run the script by using the System i Navigator Run SQL Scripts tool. Do this to display the value of the global variable. Figure 9-56 is a screen capture of the SQL script from Run SQL Scripts.

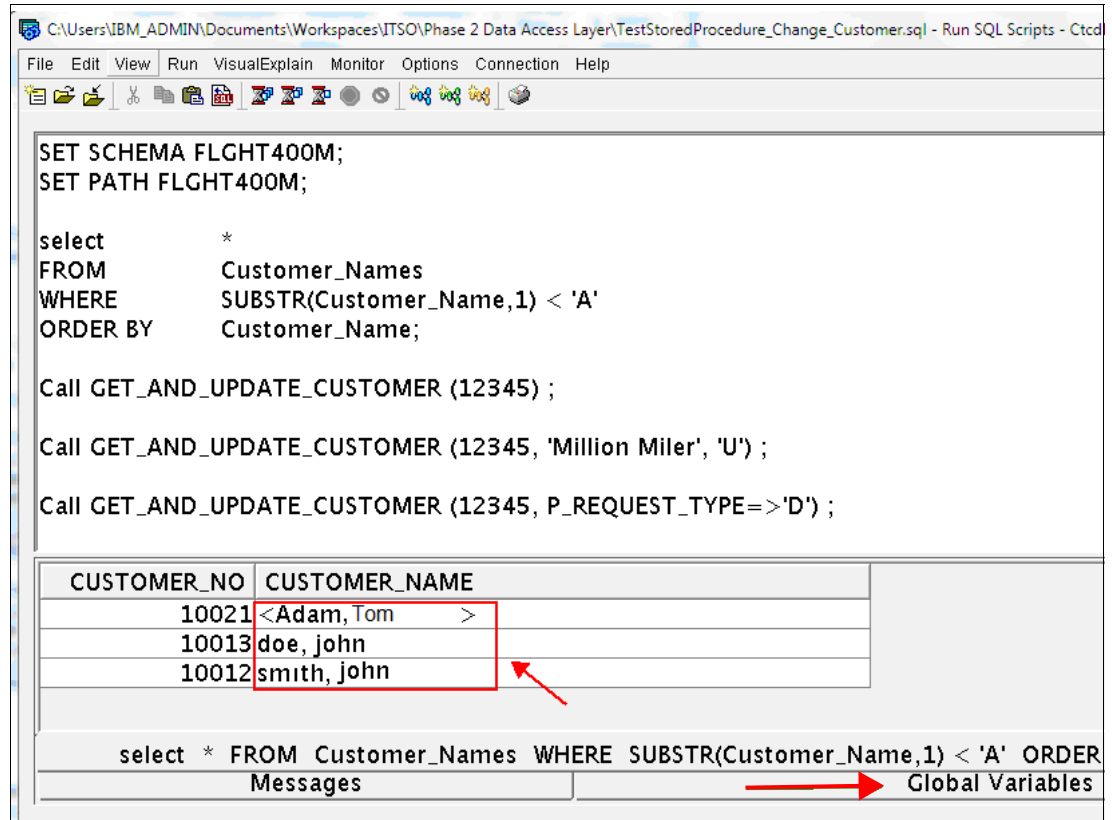


Figure 9-56 SQL script from the Navigator for i tool

Set the current schema and path to the name of the virtual layer schema FLGHT400M. Then, run a query to check for names beginning with a character less than a capital A. As you can see at the bottom of Figure 9-56, there are three of these names.

Then, call the `GET_AND_UPDATE_CUSTOMER` procedure using 10021 as the customer number and not specifying any additional parameters (R is the default). The stored procedure returns the result set confirming the incorrect customer name, as shown in Figure 9-57.

```
Call GET_AND_UPDATE_CUSTOMER (10021) ;
```

```
Call GET_AND_UPDATE_CUSTOMER (12345, 'Million Miler', 'U') ;
```

```
Call GET_AND_UPDATE_CUSTOMER (12345, P_REQUEST_TYPE=>'D')
```

CUSTOM...	CUSTOMER_NAME
10021	<Adam, Tom >

Figure 9-57 Query results after calling `GET_AND_UPDATE_CUSTOMER` with invalid data

Then, call the `GET_AND_UPDATE_CUSTOMER` procedure using 10021 as the customer number, the correct spelling of the customer name, and the update request. The procedure runs the update successfully and then displays the result set for cursor2, confirming that the change was made, as shown in Figure 9-58.

```
Call GET_AND_UPDATE_CUSTOMER (10021, 'Adam, Tom', 'U') ;
```

```
Call GET_AND_UPDATE_CUSTOMER (12345, P_REQUEST_TYPE=>'D') ;
```

CUSTOM...	CUSTOMER_NAME	LAST_CHANGED
10021	Adam, Tom	2014-02-23 10:03:25.724386

Figure 9-58 Query results after calling `GET_AND_UPDATE_CUSTOMER` with correct data

Click the **Global Variable** tab, click **Refresh**, and confirm that the `gv_When_Changed` variable contains the previous row change time stamp, as shown in Figure 9-59.

Name	Value
FLGHT400M.GV_WHEN_CHANGED	'2014-02-23-10.02.05.106275'

Messages Global Variables Call GET_AND_UPDATE_CUSTOMER (10021, 'Adam, Tom', 'U')

Figure 9-59 Updated file time stamp showing the updated file

Then, add a query to the SQL script to determine which customers currently do not have any orders. Do this by using an `EXCEPTION JOIN` between the `Customer_Names_With_Last_Changed_Info` view and the `Customer_Order` view. We determine that Tom Adam does not have any current orders. The SQL script and results are shown in Figure 9-60 on page 417.

```

select      Customer_No, T1.*
FROM        Customer_Names_With_Last_Changed_Info T1
EXCEPTION JOIN Customer_Order T2
            USING(Customer_No)
ORDER BY    Customer_Name;

```

CUSTOM...	CUSTOMER_NAME	LAST_CHANGED
10021	Adam, Tom	2014-02-23 10:03:25.724386
10022	Doe, Jane	2014-02-15 08:59:19.960572
2130	Doe, John	2014-02-15 08:59:19.960572
9862	Doe, Tom	2014-02-15 08:59:19.960572
15005	Frequent Flyer	2014-02-22 12:35:59.258267
15002	Smith, Bob	2014-02-15 08:59:19.960572
2205	Smith, Jane	2014-02-15 08:59:19.960572

Figure 9-60 SQL script and results

Call the **GET_AND_UPDATE_CUSTOMER** procedure using 10021 as the customer number and specifying the parameter name **P_REQUEST_TYPE** followed by the value for a delete operation. The usage of a named parameter to bypass optional parameters is similar to CL command support. When the procedure opens cursor2, no data is returned, as shown in Figure 9-61.

```
Call GET_AND_UPDATE_CUSTOMER (10021, P_REQUEST_TYPE=>'D') ;
```

CUSTOM...	CUSTOMER_NAME	LAST_CHANGED

Figure 9-61 Results after the record is deleted

Follow this up with a repeat of our previous query to double-check that Tom Adam was truly deleted. This is verified in Figure 9-62.

```

select      Customer_No, T1.*
FROM        Customer_Names_With_Last_Changed_Info T1
EXCEPTION JOIN Customer_Order T2
            USING(Customer_No)
ORDER BY    Customer_Name;

Call GET_AND_UPDATE_CUSTOMER (10021, P_REQUEST_TYPE=>'D') ;

```

CUSTOM...	CUSTOMER_NAME	LAST_CHANGED
10022	Doe, Jane	2014-02-15 08:59:19.96...
2130	Doe, John	2014-02-15 08:59:19.96...

Figure 9-62 Previous script call to verify that a record is gone

Summary of tasks

In our example, we combine several data-centric development techniques while taking advantage of modern tools for constructing our SQL modules. These steps are listed here:

1. Create a non-updateable SQL join view by using the **ROW CHANGE TIMESTAMP** function to return the time stamp value of the last change.

2. Create an update-only SQL view that uses the **ROW CHANGE TIMESTAMP** function on the WHERE clause to compare the current time stamp value to a global variable. This provides both optimistic locking and the inability to update data without first obtaining the current row change time stamp.
3. Create a read-only stored procedure, which returns a result set.
4. Create update and delete procedures to use the input and update only views. Use the Snippet tool to reuse code segments.
5. Create an SQL procedure that runs our update and delete procedures. The procedure captures the current row change time stamp for a given row and then passes it as a variable to the appropriate procedure.
6. Use System i Navigator Run SQL scripts to test the stored procedure. We take advantage of the Global Variable tool within the tool to view the values that are stored in our global variables

Follow this process for all re-engineering files. We accomplish all of the above without writing a single line of High Level Language (HLL) program code.

9.3.6 Creating and deploying external SQL I/O modules

Although many applications can be developed using SQL only, there are still several good reasons for using embedded SQL in a host language:

- ▶ Using SQL set processing when reading/writing large amounts of data, including automatic input/output blocking, searched updates/deletes, and so on
- ▶ Taking advantage of host language programming constructs that are not available to DB2 SQL Procedure Language (DB2 SQL PL)
 - Built-in functions not available in DB2
 - Externally described data structures
- ▶ Taking advantage of SQL functions that are not available to HLL, which include:
 - Index Only Access
 - Fast count support and aggregates through Encoded Vector Index (EVI)
 - Advanced join technology (Look ahead Predicate Generation (LPG))
 - Processor parallelism through Symmetric Multiprocessing (SMP)
 - Fast summary access through Materialized Query Table (MQT)

Installing IBM Rational Developer for i

Before developing any new HLL code (in our case, we use RPG/IV) in our example, acquire and install the IBM Rational Developer for i product. Because we already have InfoSphere Data Architect for our Database Architects and IBM Data Studio for our Database Engineers, we opt to install Rational Developer for i as part of the existing package group.

After installing and updating Rational Developer for i, view the Installed Package groups. The Database Architect appears as shown in Figure 9-63 on page 419.

Package Group Name: IBM InfoSphere Package Group Installation Directory: C:\Program Files\IBM\IDA9.1.0 Package Group Eclipse IDE: C:\Program Files\IBM\IDA9.1.0 Package Group Translations: en Package Group Architecture: 64-bit	
Packages	Features
IBM InfoSphere Data Architect Version 9.1.0.0 (9.1.0.20130604_1518) Repository Z:\Download Director\Infosphere\I901\W7_64 \INFO_DATA_ARCH_V9.1_FOR_WIN64_ML\disk1	<ul style="list-style-type: none"> ▪ SQL Routine and PL/SQL Development ▪ Cloudscape Support ▪ Core Architect Features ▪ LDM-to-XSD Transformation ▪ Netezza Support ▪ Oracle Support ▪ SQL Server Support ▪ Sybase Support ▪ Teradata Support ▪ UML-to-LDM Transformation ▪ BIRT Visual Report Designer ▪ XML Development
IBM® Rational® Developer for i Version 9.0.0.1 (9.0.1.20130801_1804) Repository http://public.dhe.ibm.com/software/awdtools/rdpower/v90/rdi/updates/	<ul style="list-style-type: none"> ▪ RPG and COBOL Tools

Figure 9-63 Database Architect information within Rational Developer for i

The IBM Data Studio plug-in appears as shown in Figure 9-64.

Package Group Name: IBM Data Studio Package Group Installation Directory: C:\Program Files\IBM\W7_DS4.1.0 Package Group Eclipse IDE: C:\Program Files\IBM\W7_DS4.1.0 Package Group Translations: en Package Group Architecture: 64-bit	
Packages	Features
IBM Data Studio client Version 4.1.0.0 (4.1.0.20130531_0821) Repository Z:\Download Director\Data Studio\I4r1\ibm_ds410_win32\disk1	<ul style="list-style-type: none"> ▪ Application Development ▪ Administration ▪ Query Tuning
IBM® Rational® Developer for i Version 9.0.0.1 (9.0.1.20130801_1804) Repository http://public.dhe.ibm.com/software/awdtools/rdpower/v90/rdi/updates/	<ul style="list-style-type: none"> ▪ RPG and COBOL Tools

Figure 9-64 IBM Data Studio client that is installed in Rational Developer for i

The Procedure wizard

We decide that all new host-centric applications access the same SQL stored procedures that are being accessed by the external applications. In Version 7.1, this is made easier by allowing host-based languages to use SQL statements to consume the result set that is defined in a stored procedure. Before this new capability, you had to use the Call Level Interface (CLI) APIs.

Here are the SQL statements that are required to enable result set consumption:

```
CALL PROCEDURE;
ASSOCIATE LOCATOR WITH PROCEDURE
ALLOCATE CURSOR FOR RESULT SET LOCATOR
FETCH FROM CURSOR
CLOSE CURSOR
```

To take advantage of result set consumption, define a template consisting of individual procedures corresponding to each SQL statement that is embedded in the RPG programs. To do this, use the RPG Procedure wizard, which is part of the RPG LPEX support in Rational Developer for i.

Start with a new empty source module named COMSUMERZT. Right-click the first blank line and select **New** → **Procedure**. Enter a procedure name that is based on the SQL function, for example, Call_Stored_Procedure. Enter a Purpose, select the **Return a value** check box, and click **Next**, as shown in Figure 9-65.

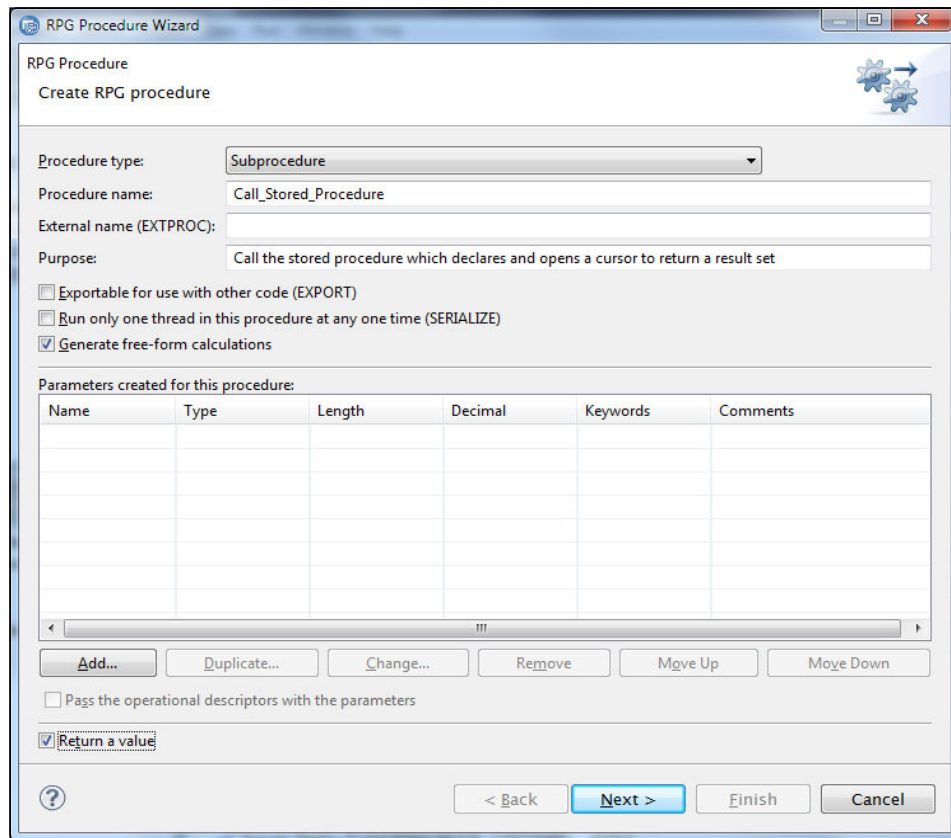


Figure 9-65 First page of the RPG procedure wizard

Choose **Indicator** as the type for the return value and click **Finish**, as shown in Figure 9-66 on page 421.

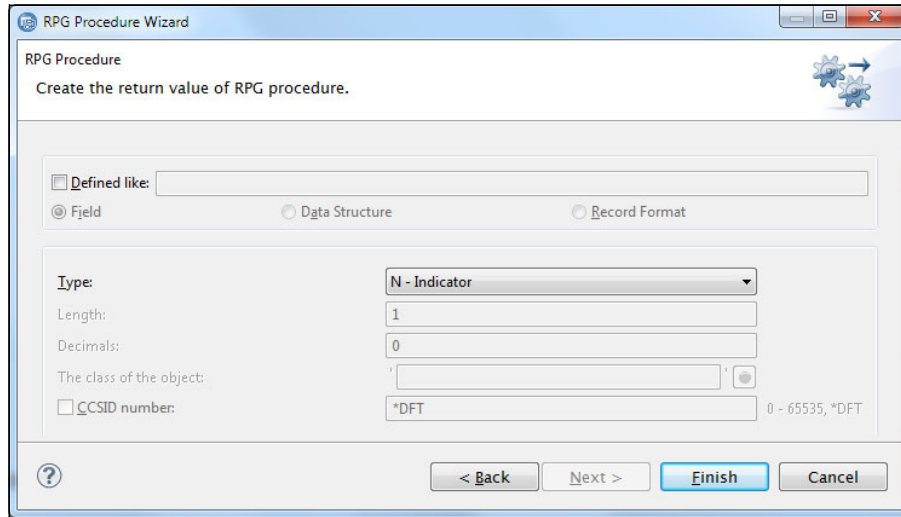


Figure 9-66 Specify Indicator and click Finish

After you click **Finish**, the following code is generated and added to the source module:

```

P*-----
P* Procedure name: Call_Stored_Procedure
P* Purpose:      Call the stored procedure that declares and opens ...
P*              a cursor to return the result set
P* Returns:     If OFF success, if ON then failure
P*-----
P Call_Stored_Procedure...
P          B
D Call_Stored_Procedure...
D          PI          N

D* Local fields
D retField      S          N

/FREE

// Your calculation code goes here

RETURN retField;

/END-FREE
P Call_Stored_Procedure...
P          E

```

Code the SQL statements that are required for the procedure and then repeat the process for all remaining SQL statements. When complete, the source module looks like Figure 9-67 (after filtering by procedure).

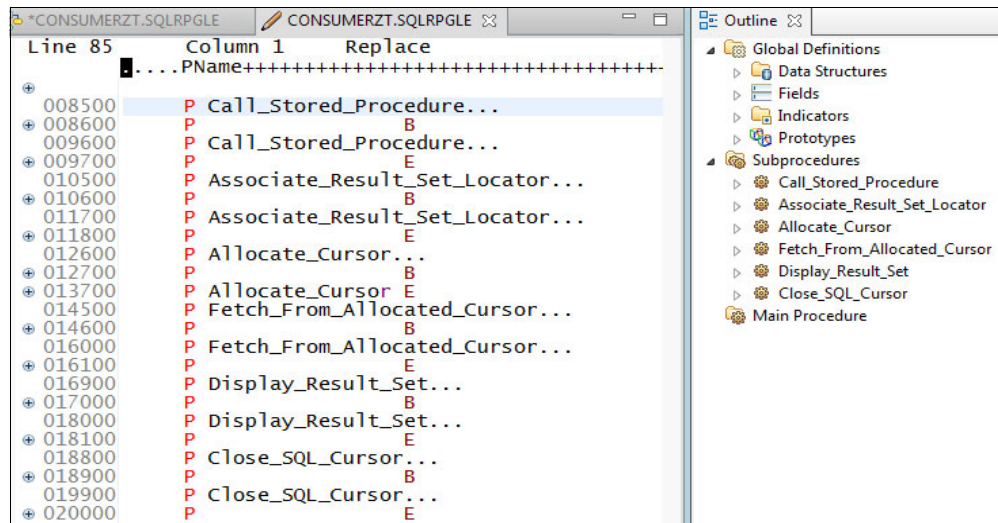


Figure 9-67 New completed source module

The Visual Application Diagramming tool

After the COMSUMERZT template is created, you can create quickly programs by copying this source member and creating a source member. In our example, we use the Visual Application Diagramming tool to provide a visual aid to new team members that are tasked with building application interface modules. The visual diagram for COMSMERRZT is shown in Figure 9-68.

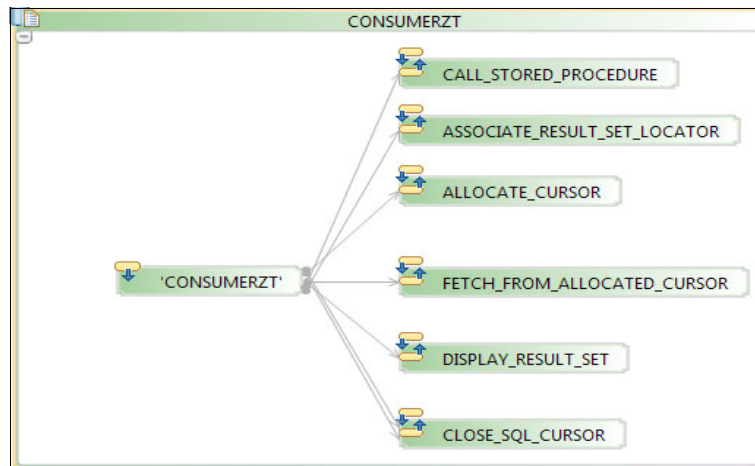


Figure 9-68 Visual diagram for COMSMERRZT

By clicking any node, Rational Developer for i opens the source member and position at the corresponding code location. In our example, we double-click the CALL_STORED_PROCEDURE and modify the following SQL CALL statement:

```

P*-----
P* Procedure name: Call_Stored_Procedure
P* Purpose:
P* Returns:
P*-----
  
```

```

P Call_Stored_Procedure...
P           B
D Call_Stored_Procedure...
D           PI

/FREE

      exec SQL call List_Customers_With_Orders (:p_StartsWith);

/END-FREE
P Call_Stored_Procedure...
P           E

```

Note: IBM provides RPG host language templates as part of its Data Centric Programming and Database Modernization workshops.

9.3.7 Dynamic SQL

During the initial phases of database re-engineering, there is little opportunity for implementing dynamic SQL I/O modules; however, as you go forward, many of your SQL I/O modules are replaced by more generic modules. The contents of section are based on an article that was published in September of 2013 by Dan Cruikshank, found at:

http://www.iprodeveloper-edition.com/iprodeveloper/september_2013?pg=32#pg32

Dynamic SQL offers a high degree of application flexibility. The SQL statement can be assembled based on parameters that are received from application interfaces such as RPG Open Access. In fact, many customers have applications in place today that build OPNQRYF statements. In essence, OPNQRYF is a primitive form of creating dynamic queries and consuming the results in a host-centric program. These applications are prime candidates for conversion to dynamic SQL through Open Access.

Types of dynamic SQL

There are two types of dynamic SQL. The first is referred to as fixed-list SQL. For example, consider the following SQL statement:

```

select *
FROM Customer_Names_With_Last_Changed_Info
WHERE SUBSTR(Customer_Name,1) < 'A'
ORDER BY Customer_Name;

```

The * implies that the column list is fixed based on the format of the Customer_Names_With_Last_Changed_Info SQL view.

The second type of SQL is referred to as varying-list SQL. Unlike the previous example, the column list varies from one SQL statement to the other. This is shown in the following SQL statements:

```

SELECT customer_no, customer_name
FROM CUSTOMER_NAMES WHERE customer_no > 15005;
SELECT customer_no, customer_name, tickets_ordered
FROM Customers_With_Orders WHERE tickets_ordered > 1;

```

A fixed list SQL statement does not require a descriptor. A varying list SQL statement does require a descriptor. Both types of statements require descriptors if the WHERE clause contains parameter markers.

Parameter markers

A parameter marker is a placeholder for a host variable that is contained within the SQL text string. The parameter marker is replaced at execution time by adding the USING or INTO clause with the appropriate SQL statement. For a WHERE clause, the USING clause is added to the **OPEN** statement like this:

```
sql_String = 'SELECT * FROM '  
            + %TrimR(:sqlTableName)  
            + ' WHERE class = ? AND tickets_ordered > ?' ;  
PREPARE statement_name FROM :sql_String;  
DECLARE cursor_name CURSOR FOR statement_name;  
OPEN cursor_name USING :v_class , :v_tickets_ordered;
```

The host variables v_class and v_tickets_ordered must be listed and correspond to the parameter marker position within the SQL text string.

SQL descriptors

So where does the descriptor come in? Consider the fact that the WHERE clause can change depending on the search argument (for example, WHERE class > ? AND tickets_ordered = ?, WHERE class = ?, and WHERE agent_no = ? AND class > ? AND flight_number = ?). Without a descriptor, you must code multiple SQL **OPEN** statements for each possible combination. This can become a maintenance challenge.

To avoid this situation, use the **DESCRIBE INPUT USING DESCRIPTOR** SQL-descriptor-name (or **INTO descriptor-name** if you are using an **SQLDA**) immediately following the **PREPARE SQL** statement as shown below:

```
sql_String = 'SELECT * FROM '  
            + %TrimR(:sqlTableName)  
            + ' WHERE tickets_ordered > ? AND agent_no = ?' ;  
PREPARE statement_name FROM :sql_String;  
DECLARE cursor_name FOR statement_name;  
ALLOCATE DESCRIPTOR dynamic_where_clause MAX(:v_parms)  
DESCRIBE INPUT USING DESCRIPTOR dynamic_where_clause;  
For i to :v_parms;  
SET DESCRIPTOR values;  
EndFor;  
OPEN cursor_name USING SQL DESCRIPTOR dynamic_where_clause;
```

9.3.8 Bridging existing programs to SQL service programs

Like many IBM i customers, the Flight/400 heritage applications, which are written in the IBM RPG programming language, are still relied upon to run the business while the reengineering effort is in progress. These RPG programs must access the SQL procedures that are serving the new SQL views. Duplication of rules and logic can (and does) result in expensive dual maintenance and multiple versions of the truth.

Rational Open Access: RPG Edition (ROA), introduced in Version 7.1 and retrofitted to Version 6.1, can eliminate the high cost of dual maintenance and expensive program rewrites by allowing RPG I/O operations to be intercepted and transformed to access SQL procedures. A new keyword (**HANDLER**) can now be added to an existing RPG file specification.

The handler keyword specifies either a program or service program entry point that is called by the IBM i OS. This new support can substantially reduce the amount of heritage code rewriting that is required to convert existing programs to share the new SQL modules.

Figure 9-69 shows a high-level view of “Isolation” and Shared Result Set.

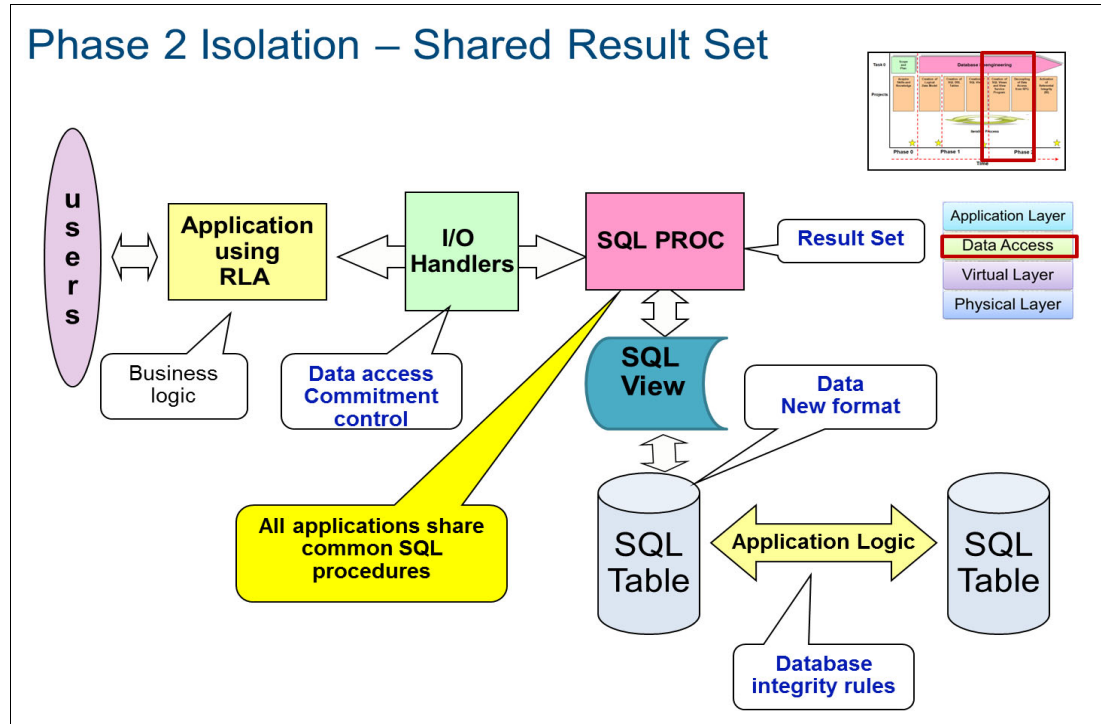


Figure 9-69 Phase 2 Isolation - Shared Result Set

The handler program transforms the RPG traditional “record at a time” I/O operation to the appropriate SQL stored procedure call. The handler program consumes result sets in the case of read operations or calls the same INSERT, UPDATE, or DELETE SQL stored procedures that are accessed by new and external applications.

Figure 9-70 shows a visual diagram for an RPG handler program transforming an RPG write to a dynamic SQL procedure call.

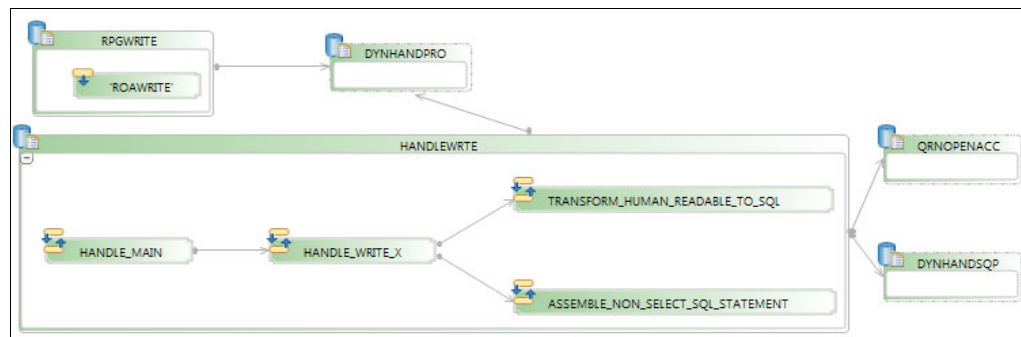


Figure 9-70 Visual diagram for the RPG Open Access Handler

The RPG program issuing the **WRITE** operation shares a common include module (DYNHANDPRO) with the handler program. The handler program intercepts the **WRITE** operation using the names values structure that is passed in the QRNOOPENACC. The handler constructs the non-Select statement and passes it to the dynamic SQL service program.

During the initial phases of database re-engineering, there is no opportunity for implementing handler programs; however, as you go forward, many of the existing RPG programs are modified to use handler bridge programs. The following paragraphs are based on an article that was published in September 2013 by Dan Cruikshank, found at:

http://www.iprodeveloper-edition.com/iprodeveloper/september_2013?pg=32#pg32

An input descriptor is also used to define the variables for non-Select SQL statements (that is, **INSERT**, **UPDATE**, **DELETE**, and **CALL**). Here are examples of non-Select SQL statements where parameter markers are used instead of literal values:

```
INSERT INTO CUSTOMER_ORDERS VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?);
UPDATE CUSTOMER_ORDERS SET (class, flight_number, departure_date) = (?,?,?)
  WHERE customer_number = ?;
DELETE FROM CUSTOMER_NAME WHERE Customer_No = ?;
CALL ADD_NEW_ORDER (?,?,?,?,?,?,?,?);
```

Here is the non-select SQL statement process:

```
sql_String = 'INSERT INTO CUSTOMER_ORDERS VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?)';
PREPARE statement_name FROM :sql_String;
ALLOCATE DESCRIPTOR non_select_SQL MAX(:v_parms)
DESCRIBE INPUT USING DESCRIPTOR non_select_SQL;
For i to :v_parms;
  SET DESCRIPTOR values;
EndFor;
EXECUTE statement_name USING DESCRIPTOR non_select_SQL.
```

RPG does not provide all of the information that is needed to build an SQL statement in one operation, and RPG does not describe how the file is opened. There is no one-to-one correlation between an RPG I/O operation and an SQL statement. This means that you must build your SQL statements using an assembly type process where each subassembly corresponds to an SQL clause. Some of the information is available at open time; the rest becomes available when the first I/O operation is performed. After all of the information is obtained from RPG, you can produce the final assembly or SQL full-select statement and then run that statement.

Table 9-6 contains a list of the RPG subfields passed to the handler (through the QRNOOPENACC data structure), which are used as input to construct the appropriate SQL clause.

Table 9-6 RPG subfields that are passed to the Open Access handler

Element	Provided at	RPG operation	Used to construct	Notes
compileFile, externalFile, externalMember	Open	Implicit/Explicit OPEN	FROM table-name, INSERT INTO table-name, UPDATE table-name	Priority order: Member, external File, and compile file.
keyed	Open	Position 19 of F-spec contains a 'K'.	ORDER BY-Clause	RPG does not provide ordering keys. Must be provided as user option or use API.

Element	Provided at	RPG operation	Used to construct	Notes
namesValue	First IO	UPDATE and WRITE	Table Descriptor functions, column, and parameter lists on INSERT and UPDATE	SELECT * can be used for read operations; implicit column list can be used for INSERT .
keyNamesValue	First IO	SETxx, CHAIN, READE, READPE, UPDATE, and DELETE	SELECT-clause. WHERE-clause, ORDER BY clause, and Input Descriptor	These columns can be used on the ORDER BY clause.
userArea.Where_Clause	First IO	CHAIN, READ, READE, READP, READPE, UPDATE, and DELETE	WHERE-clause	Overrides keyNamesValues.
userArea.OrderBy_Clause	First IO	CHAIN, READ, READE, READP, and READPE	ORDER BY clause	Overrides API.

9.3.9 Activating referential integrity

As a preferred practice, create, test, and deploy the Phase 2 stored procedures in to production before attempting to use RI.

The final process of the IBM Database Modernization Strategy is the implementation of RI. This is referred to as Phase 3. In essence, when you reach this phase, your candidate tables are created by using SQL DDL. In addition, all access is now being done directly against the views or indirectly through stored procedures (SQL or external).

Figure 9-71 shows the main goals of Phase 3 - Integrity.

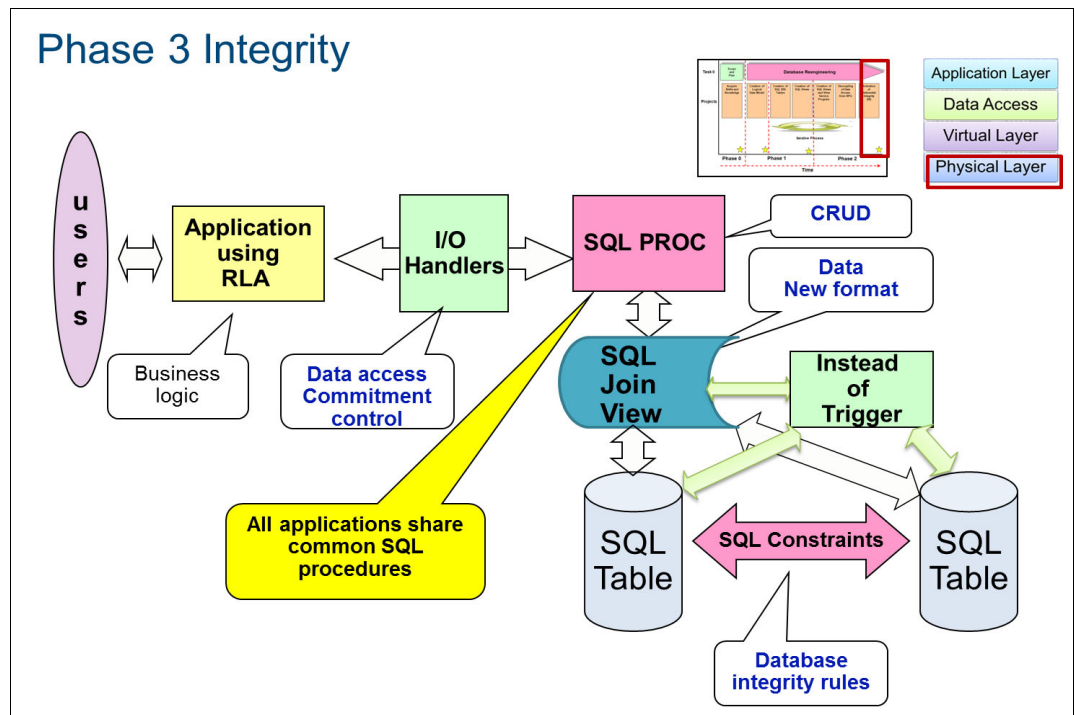


Figure 9-71 Phase 3 - Integrity

The Flight/400 programs use application-centric rules to implement data integrity by doing a combination of SETLL to determine existence and then running the appropriate **WRITE**, **UPDATE**, or **DELETE** operation. Program joins are performed by a **READ** or **READE** followed by one or more **CHAIN** operations. This type of behavior required the duplication of unique keys from parent or master tables to child or dependent tables.

These back door approaches might have been acceptable when the application was developed for back-office operations, but with the advent of the internet and the proliferation of data piracy, these techniques are no longer considered good practices.

In the new data model, all relationships are explicitly defined using declarative RI constraints. The Primary-Foreign key columns that are used in the constraint are used to join the new tables together.

Creating new physical data models

Before continuing with Phase 3 in our example, we decide to create two PDMs representing the physical and virtual layers of the Flight/400 database. We accomplish this task by using the same reverse engineering process that we used in Phase 1 with some variations.

For the physical layer, create a PDM named Physical Data Layer, choose schema FLIGHT_DB2, select the **Tables, Indexes, Triggers, User-defined types,** and **Sequences** check boxes, and click **Finish**, as shown in Figure 9-72. After the physical layer is created, Figure 9-73 shows the expended PDM.

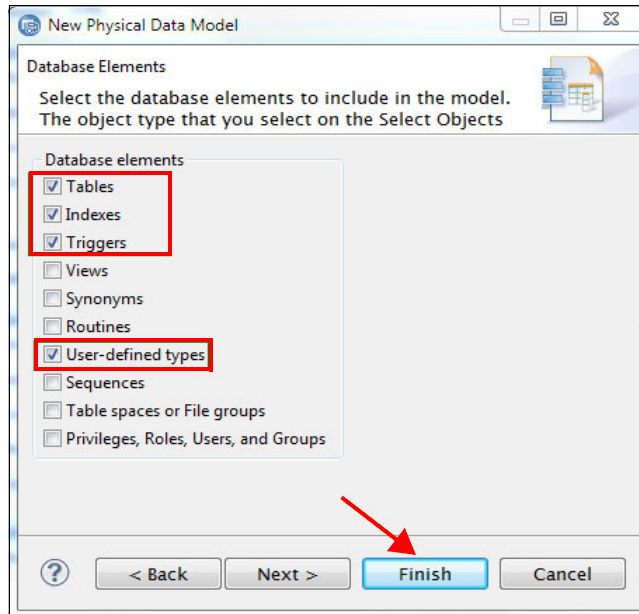


Figure 9-72 Physical Data Layer PDM elements

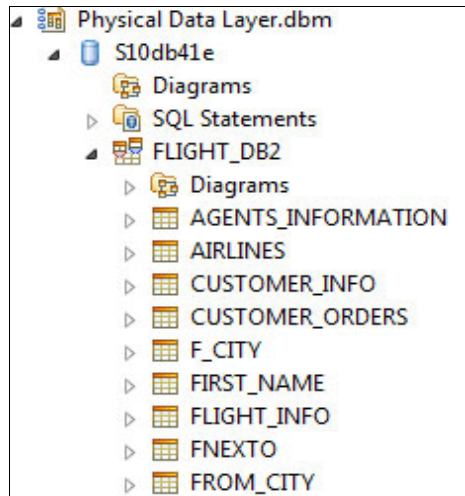


Figure 9-73 Expanded PDM for Physical Data Layer PDM

For the virtual layer, create a PDM named Virtual Data Layer, choose schema FLIGHT400M, and select the **Views**, **Synonyms**, and **Routines** check boxes, as shown in Figure 9-74. After the virtual layer is created, Figure 9-75 shows the expended PDM.

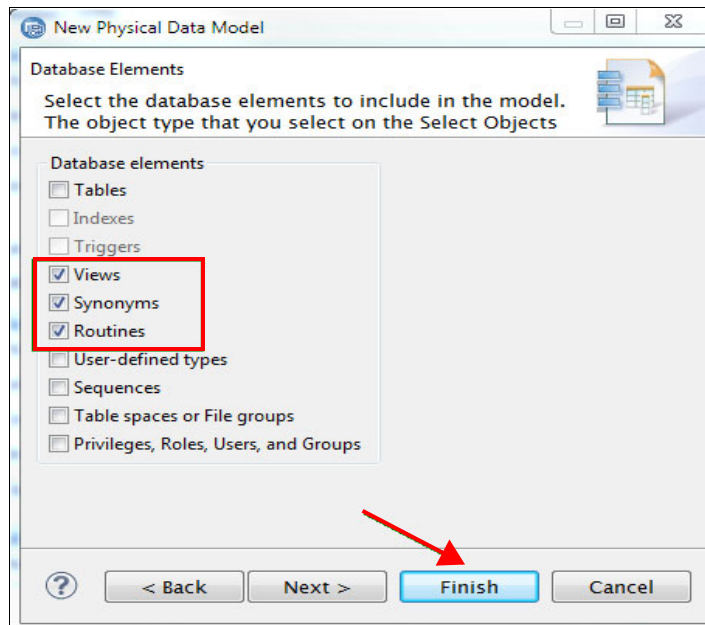


Figure 9-74 Virtual Data Layer PDM elements

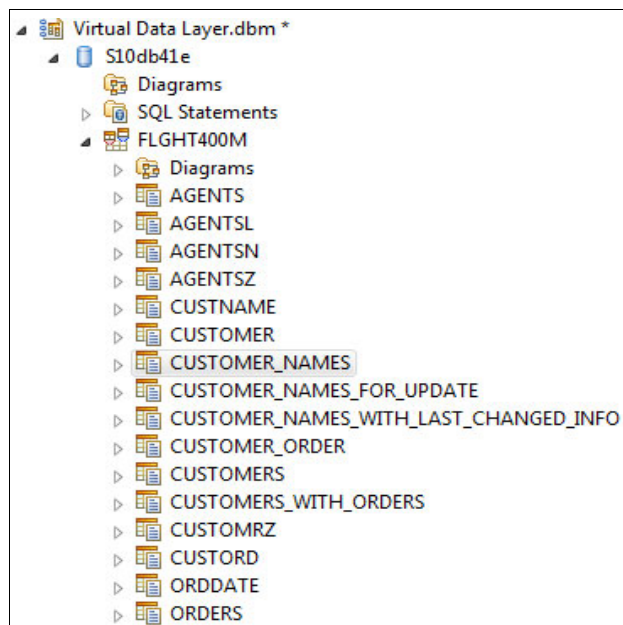


Figure 9-75 Expanded Virtual Data Layer PDM

Note: Although many data elements appear in the data model list, they cannot be modified from the list. You must retain the original data development project for future modifications.

Preparing for referential integrity

The Flight/400 application consists of many files (for example, Agents, Airlines, Flights, and Customers). Obviously, there are relationships and dependencies between these files. Associated with these relationships are business rules, which define what data must exist in order for a relationship to be created or terminated. For example, a customer cannot enter an order for a flight whose flight number does not exist in the Flights file. A customer cannot be deleted if the customer has open orders. However, a flight can be canceled even though it is referenced by an order.

These types of rules might exist in application program code but they cannot be enforced unless all data access is restricted to these programs. This is not likely because new applications using modern interfaces are being added to the Flight/400 portfolio.

By establishing primary keys as part of the new table in Phase 1, we provide a solution to the previously stated issues. In combination with a foreign key, we can now put the business rules in one place: the database. All applications, regardless of language or interface, now use the same rules.

DB2 for i provides several means for enforcing business rules at the database level. This includes constraints, routines (functions, triggers, and procedures), views, and so on. All of these techniques are taught in the Data-Centric Programming workshop.

We chose initially to focus on data integrity rules, which can be defined as constraints. There are four types of constraints, each with its own capabilities. Table 9-7 describes the constraints and their purposes.

Table 9-7 Constraints and purposes

Constraint	Purpose
UNIQUE	Forbids duplicate values within one or more columns.
PRIMARY KEY	Same as UNIQUE; however, NULL values are not allowed.
FOREIGN KEY	Also known as a referential constraint. Defines the logical rules that are imposed on INSERT , UPDATE , and DELETE operations to ensure integrity between data relationships.
CHECK	A logical rule that ensures data integrity at the column level

We use the diagram that is shown Figure 9-75 on page 430 as a blueprint for how we implement our data integrity rules in our example.

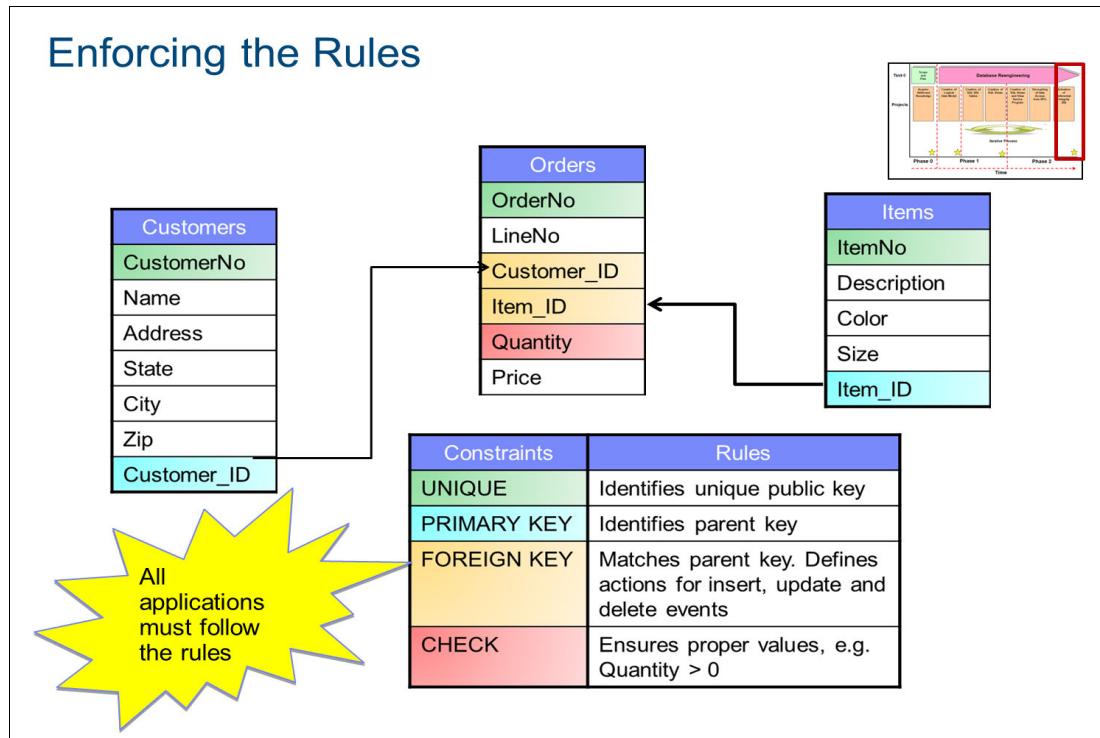


Figure 9-76 Constraints and enforcement of rules

Based on the blueprint, one of the goals is to eliminate duplication of non-primary key columns. We no longer require unique keys such as Customer_No to be in more than one table.

Here are the steps to take to add a RI constraint between Customer_Info and Customer_Orders:

1. Identify and delete orphans from the Customer_Orders table.
2. Alter Customer_Orders by performing the following tasks:
 - Dropping column Customer_No
 - Adding column Customer_ID
3. Alter Customer_Orders by adding a foreign key constraint.
4. Populate the new Customer_ID column in Customer_Orders.
5. Remove Customer_No column references from all views that were built over Customer_Orders.
6. Add Instead of triggers to update views.
7. Test existing procedures to ensure that no harm was done.

Identifying and deleting orphans

To avoid constraint violations, write queries to determine whether any orders contain customer numbers that do not exist in the customers file. In our example, we write the following query to determine how many orders do not have a valid customer number in the original source data:

```
SELECT count(*)
```

```

FROM FLGHT400.ORDERS T1
EXCEPTION JOIN FLGHT400.CUSTOMERS
  USING (CUSTOMER_NO);

```

Based on the results of the preceding query, there are 207 orphan rows in the orders file. We then write the following query to determine how many orders there are per customer:

```

SELECT CUSTOMER_NO, count(*) Total_Orphans
FROM FLGHT400.ORDERS T1
EXCEPTION JOIN FLGHT400.CUSTOMERS
  USING (CUSTOMER_NO)
GROUP BY CUSTOMER_NO
ORDER BY Total_Orphans DESC;

```

Based on the results of the preceding query, there are only three invalid customer numbers. Because we are working with test data, we decide to delete the orphans. Use the following SQL **DELETE** statement to delete the orphans from the new CUSTOMER_ORDERS table:

```

DELETE FROM FLIGHT_DB2.CUSTOMER_ORDERS
WHERE CUSTOMER_NO NOT IN (
SELECT CUSTOMER_NO
FROM FLIGHT_DB2.CUSTOMER_INFO T1) ;

```

Note: The decision to handle orphans typically is made by the business after further analysis.

Dropping and adding columns to tables

Create a diagram in the Physical Layer data model that is named Phase 3 Customer Orders with Rational Developer for i. Drag CUSTOMER_INFO and CUSTOMER_ORDERS on to the blank diagram, as shown in Figure 9-77.

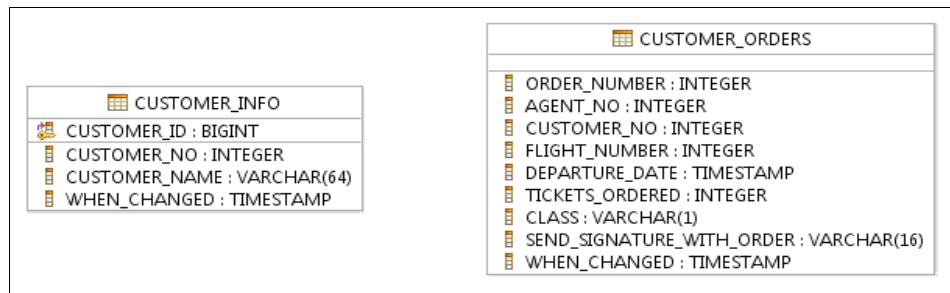


Figure 9-77 Drag CUSTOMER_INFO and CUSTOMER_ORDER to the diagram view

Dropping an existing column from the data model

Click table **CUSTOMER_ORDERS** and select **Columns** from the Properties view. Highlight **CUSTOMER_NO** and delete it by clicking the **Delete** symbol, as shown in Figure 9-78.

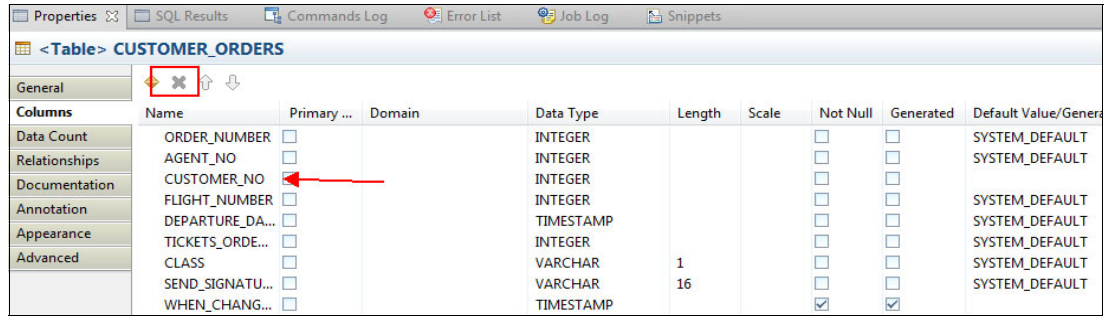


Figure 9-78 Deleting CUSTOMER_NO from data model

The CUSTOMER_NO column is no longer an attribute of CUSTOMER_ORDERS. The diagram is updated to look like Figure 9-79.

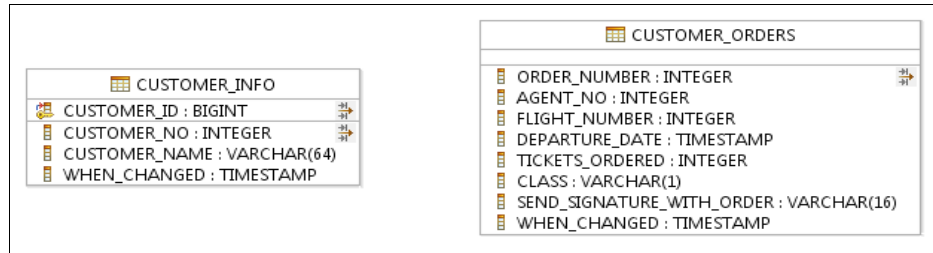


Figure 9-79 Updated diagram with CUSTOMER_NO removed from CUSTOMER_ORDERS

Adding a foreign key constraint

We use the Palette tool to add an identifying *one-to-many* declared relationship between CUSTOMER_INFO and CUSTOMER_ORDERS. The tool automatically propagates the CUSTOMER_ID primary key column as a foreign key column in the CUSTOMER_ORDERS table, as shown in the updated diagram in Figure 9-80.

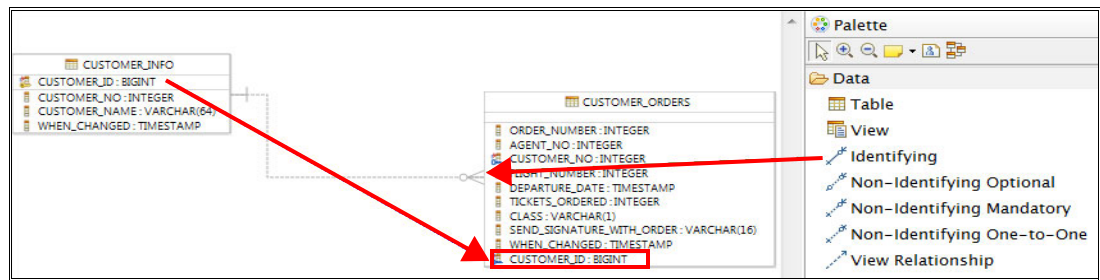


Figure 9-80 Adding a foreign key constraint

We add documentation to the constraint properties to identify the parent-child relationship. We also validate the constraint name and integrity rules. Our complete diagram is shown in Figure 9-81 on page 435.

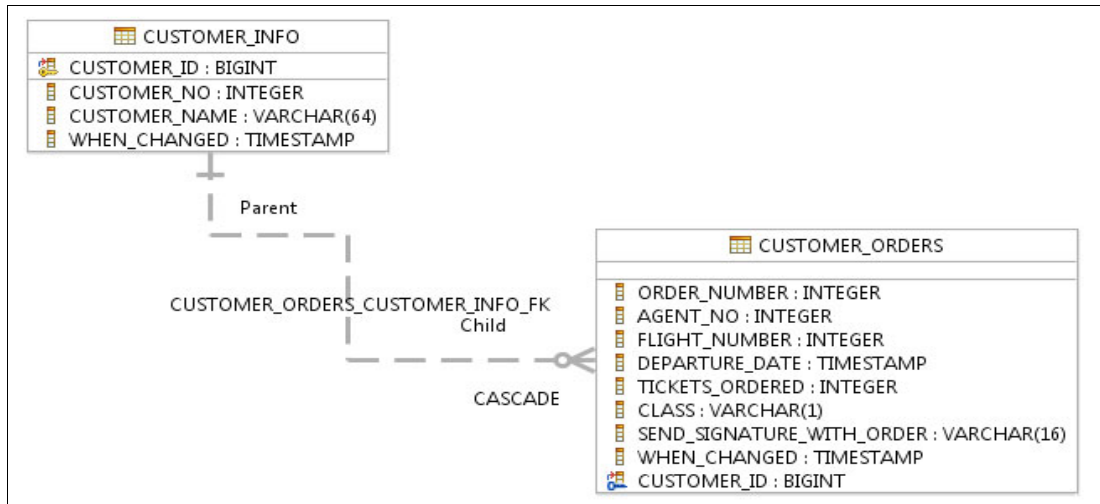


Figure 9-81 Completed diagram

Generating the ALTER table SQL DDL statements

One of the strengths of both InfoSphere Data Architect and IBM Data Studio is their ability to compare the graphical data model to the actual database, produce a side-by-side comparison, and ultimately generate the DDL differences (either as **ALTER** or **CREATE** statements).

You can accomplish this task by right-clicking the schema name (Flight_DB2 in our example) that is associated with the physical data layer data model and selecting **Compare With** → **Original Source** (DB2 for i in our case), clearing **Views**, and clicking **Finish** on the Filtering Options window to open the Structural Compare panels.

The tables that we change appear in both the graphical data model (highlighted in blue) and the DB2 for i database (highlighted in green), as shown in Figure 9-82.

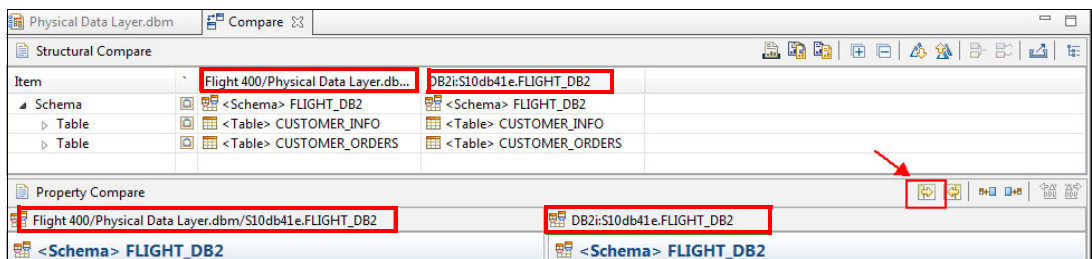


Figure 9-82 Comparing with the original source view

Click the **Copy** from left to right button indicated by the red arrow in Figure 9-82 on page 435. This action replaces the resource on the right with the changes in the resource on the left. It prepares the resource for DDL generation. Then, click **Generate Delta DDL**. Clear quoted identifiers and click **Next** twice. We name the generated DDL script CompareDDL_Customer_Orders_With_RI, select **Open DDL file for editing**, and click **Finish**, as shown in Figure 9-83.

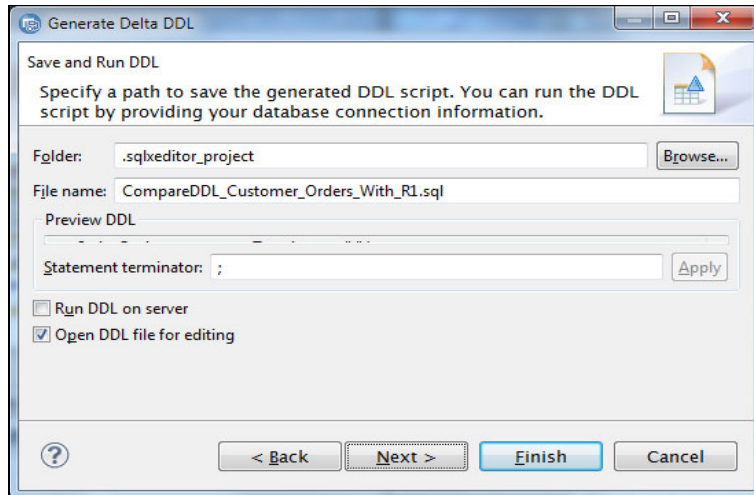


Figure 9-83 Generate Delta DDL wizard

The generated SQL script includes DDL statements for dropping and re-creating the SQL views built over CUSTOMER_ORDERS. We decide to do task separately. We also want to update the new CUSTOMER_ID foreign key column before dropping the CUSTOMER_NO column from CUSTOMER_ORDERS.

We modify the generated DDL script by adding an **UPDATE** statement following the addition of the foreign key column, as shown here:

```
ALTER TABLE FLIGHT_DB2.CUSTOMER_ORDERS ADD COLUMN CUSTOMER_ID FOR COLUMN CUST_ID
BIGINT NOT NULL DEFAULT IMPLICITLY HIDDEN;
```

```
UPDATE FLIGHT_DB2.CUSTOMER_ORDERS U1
  SET CUSTOMER_ID =
    (SELECT CUSTOMER_ID
     FROM FLIGHT_DB2.CUSTOMER_INFO T1
     WHERE U1.CUSTOMER_NO = T1.CUSTOMER_NO)
WHERE U1.CUSTOMER_NO IN
(SSELECT CUSTOMER_NO
 FROM FLIGHT_DB2.CUSTOMER_INFO T1);
```

Note: Adding a column causes DB2 for i to make a copy of the data. This copy can be avoided if the foreign columns are added as part of Phase 1. This is known as the “Measure twice, cut once” method. The time savings in eliminating copies might justify the expense of having a good logical data modeling tool that identifies implicit relationships.

The following SQL statement adds the referential constraint:

```
ALTER TABLE FLIGHT_DB2.CUSTOMER_ORDERS ADD CONSTRAINT
FLIGHT_DB2.CUSTOMER_ORDERS_CUSTOMER_INFO_FK FOREIGN KEY
(CUSTOMER_ID)
REFERENCES FLIGHT_DB2.CUSTOMER_INFO
```

```
(CUSTOMER_ID)
ON DELETE CASCADE;
```

Remove the **DROP** and **CREATE VIEW** statements because these statements are done separately. Below is the final statement in the script. It removes the customer number column from the customer orders table and, by virtue of the **CASCADE** option, drop any views, indexes, or constraints that reference the **CUSTOMER_NO** column:

```
ALTER TABLE FLIGHT_DB2.CUSTOMER_ORDERS DROP COLUMN CUSTOMER_NO CASCADE;
```

Note: The DB2 for i **ALTER TABLE** statement can combine many actions in a single step. These steps might be more efficient than manually dropping and re-creating all dependent objects that are created over the altered table.

For more information about the usage of **ALTER TABLE**, see the following website:

<http://pic.dhe.ibm.com/infocenter/iseriess/v7r1m0/topic/db2/rbafzatab1.htm>

Refactoring SQL views

We create a diagram that is named Phase 3 Customers with Views in the virtual layer model showing the current state of the SQL tables and views. The diagram is shown in Figure 9-84.

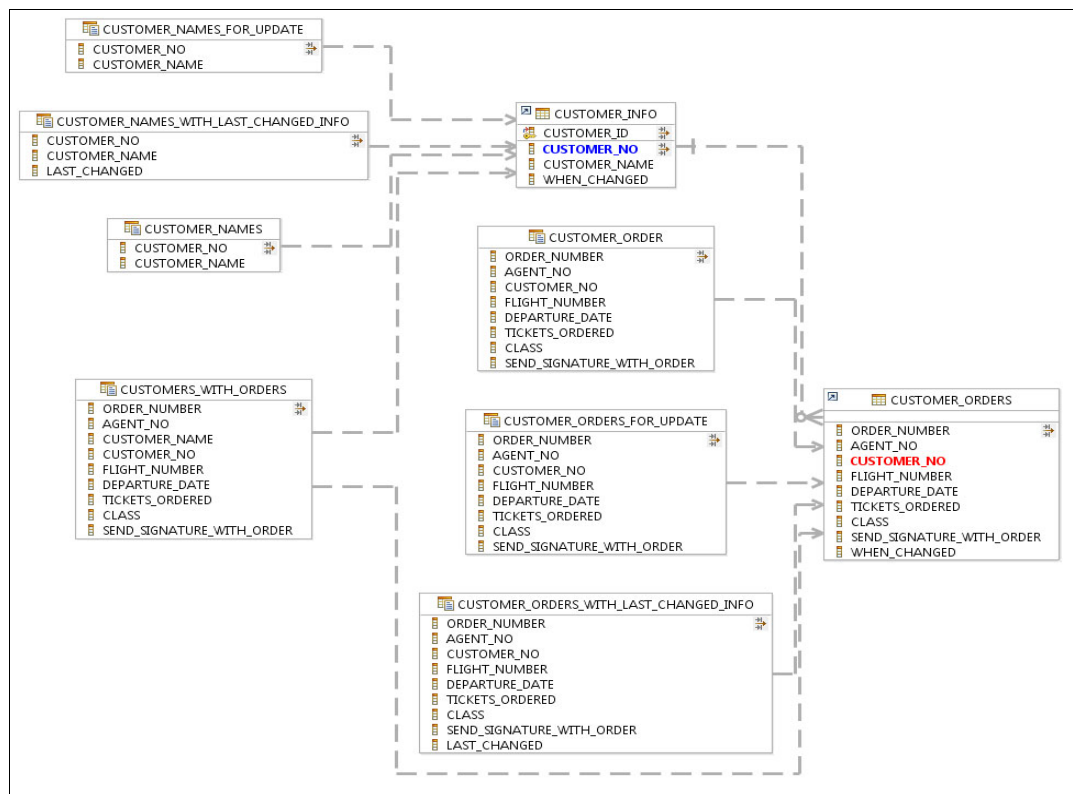


Figure 9-84 Phase 3 Customers with Views diagram

We first identify all SQL view references to the Customer_Orders table. We then check for references to the Customer_No column. If the view was a join including Customer_Info and the projected result reference was to the Customer_Info table, we highlight the view name in blue signifying that the view might require changes if it is joining on Customer_No. All other view names were highlighted in red indicating the view must be changed, as shown in Figure 9-85.

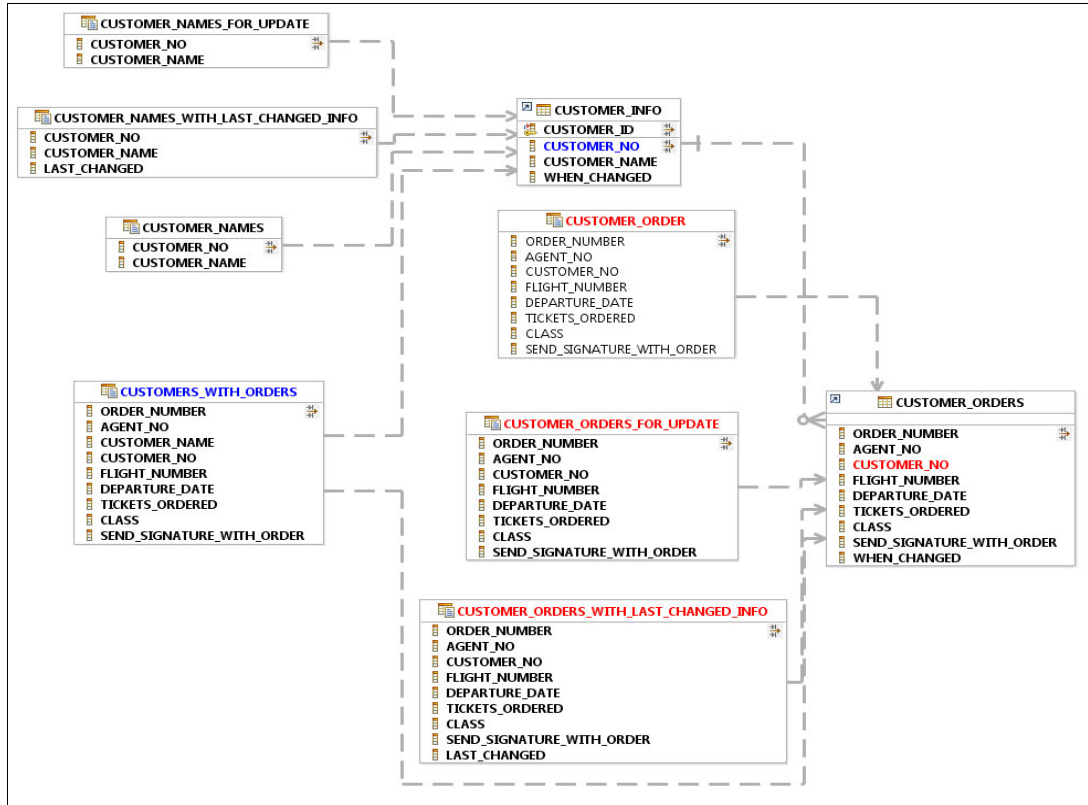


Figure 9-85 Updated Phase 3 diagram

We then modify the SQL statement in the red views as follows:

1. We add join syntax referencing the Customer_Info table.
2. If the SQL statement was using **SELECT ***, replace the ***** with all column names from Customer_Orders. We qualify Customer_No with correlation name T1.

We modify the SQL statement in the blue views to use the primary key as the join column.

We use the compare utility to compare the changed views to the IBM i DB2 schema. The views do not have a matching resource on IBM i because the views were deleted when the column Customer_No was dropped from table Customer_Orders. We choose to copy the resources to the right and generate the DDL statements.

We modify the generated **CREATE VIEW** statements to use **CREATE OR REPLACE VIEW**. Here is the generated DDL script with modifications (changes to the SQL statement are shown in **bold**):

```
CREATE OR REPLACE VIEW FLGHT400M.CUSTOMER_ORDER AS
SELECT ORDER_NUMBER, AGENT_NO, T1.CUSTOMER_NO, FLIGHT_NUMBER, DEPARTURE_DATE,
TICKETS_ORDERED, CLASS, SEND_SIGNATURE_WITH_ORDER
FROM FLIGHT_DB2.CUSTINFO T1
JOIN FLIGHT_DB2.CUSTORDERS T2 ON T1.CUSTOMER_ID = T2.CUSTOMER_ID;
```

```

RENAME FLGHT400M.CUSTOMER_ORDER TO SYSTEM NAME CUSTORDER;

CREATE OR REPLACE VIEW FLGHT400M.CUSTOMER_ORDERS_FOR_UPDATE AS
SELECT ORDER_NUMBER, AGENT_NO, T1.CUSTOMER_NO, FLIGHT_NUMBER, DEPARTURE_DATE,
TICKETS_ORDERED, CLASS, SEND_SIGNATURE_WITH_ORDER
FROM FLIGHT_DB2.CUSTINFO T1
JOIN FLIGHT_DB2.CUSTORDERS T2 ON T1.CUSTOMER_ID = T2.CUSTOMER_ID
WHERE ROW CHANGE TIMESTAMP FOR T1 = FLGHT400M.GV_WHEN_CHANGED;

RENAME FLGHT400M.CUSTOMER_ORDERS_FOR_UPDATE TO SYSTEM NAME CUSTORDRSU;

CREATE OR REPLACE VIEW FLGHT400M.CUSTOMER_ORDERS_WITH_LAST_CHANGED_INFO AS
SELECT T1.ORDER_NUMBER, T1.AGENT_NO, T3.CUSTOMER_NO, T1.FLIGHT_NUMBER,
T1.DEPARTURE_DATE,
T1.TICKETS_ORDERED, T1.CLASS, T1.SEND_SIGNATURE_WITH_ORDER, ROW CHANGE TIMESTAMP
FOR T2 AS LAST_CHANGED
FROM (FLIGHT_DB2.CUSTOMER_ORDERS T1
JOIN FLIGHT_DB2.CUSTOMER_ORDERS T2
ON T1.CUSTOMER_ID = T2.CUSTOMER_ID AND T1.WHEN_CHANGED = T2.WHEN_CHANGED)
JOIN FLIGHT_DB2.CUSTINFO T3 ON T1.CUSTOMER_ID = T3.CUSTOMER_ID;

RENAME FLGHT400M.CUSTOMER_ORDERS_WITH_LAST_CHANGED_INFO TO SYSTEM NAME CUSORDWRCT;

CREATE OR REPLACE VIEW FLGHT400M.CUSTOMERS_WITH_ORDERS
(ORDER_NUMBER FOR COLUMN ORDER00001, AGENT_NO, CUSTOMER_NAME FOR COLUMN
CUST000001, CUSTOMER_NO FOR COLUMN CUST000002, FLIGHT_NUMBER FOR COLUMN
FLIGH00001, DEPARTURE_DATE FOR COLUMN DEPAR00001, TICKETS_ORDERED FOR COLUMN
TICKE00001, CLASS, SEND_SIGNATURE_WITH_ORDER FOR COLUMN SEND_00001) AS
SELECT ORDER_NUMBER, AGENT_NO, CUSTOMER_NAME, T1.CUSTOMER_NO, FLIGHT_NUMBER,
DEPARTURE_DATE,
TICKETS_ORDERED, CLASS, SEND_SIGNATURE_WITH_ORDER
FROM FLIGHT_DB2.CUSTINFO T1 JOIN FLIGHT_DB2.CUSTORDERS T2 ON T1.CUSTOMER_ID =
T2.CUSTOMER_ID;

RENAME FLGHT400M.CUSTOMERS_WITH_ORDERS TO SYSTEM NAME CUSTWORDRS;

```

Adding an INSTEAD Of trigger to the view

As the database becomes more normalized, more SQL join views are required to de-normalize the data for selection purposes. Although not an issue for read-only scenarios, it can become an issue for insert, update, or delete transactions that are being performed against the de-normalized table. The INSTEAD OF trigger (IOT) provides a solution, allowing existing programs to continue to work without requiring code changes.

An INSTEAD OF trigger enables the usage of update, delete, or insert operations to be performed against an SQL view, which is considered to be non-updateable, non-deleteable, and non-insertable. A join view is one such view.

An **UPDATE**, **DELETE**, or **INSERT** performed against a READ-ONLY view fails unless an INSTEAD OF trigger is attached to the view. The update, delete, or insert is detoured and the work is done by the IOT.

Figure 9-86 shows the Phase 3 Customer Orders with Views diagram.

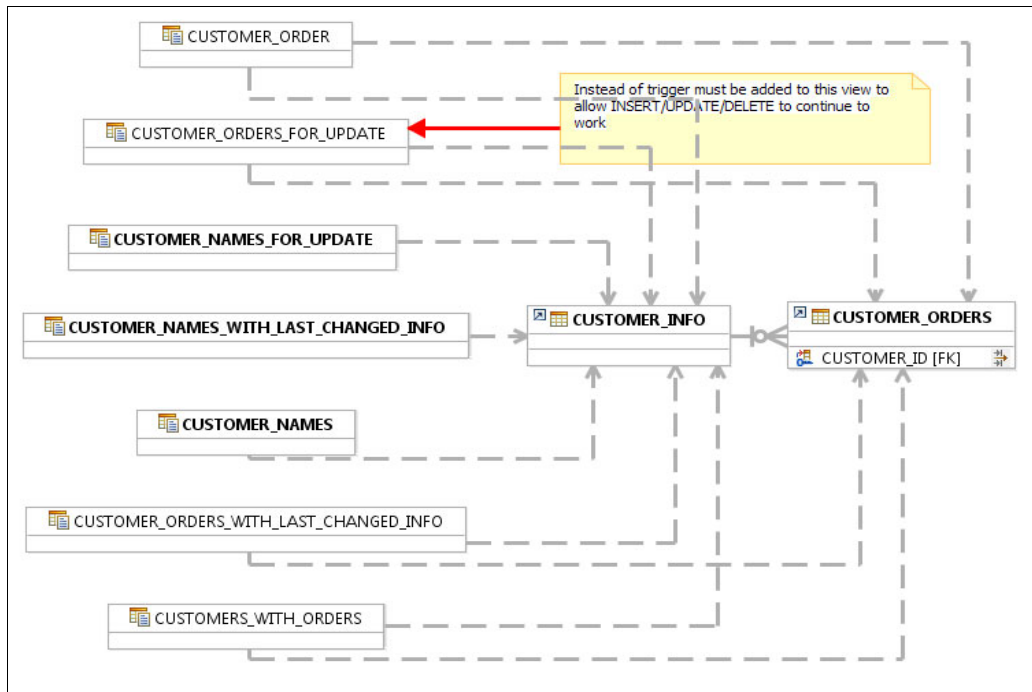


Figure 9-86 Phase 3 Customer Orders with Views diagram

We create a script that is named

CreateInsteadOfTriggers_For_Customer_Orders_For_Update and enter the following SQL **CREATE TRIGGER** statements:

```
CREATE OR REPLACE TRIGGER CANCEL_CUSTOMER_ORDER
INSTEAD OF DELETE ON CUSTOMER_ORDERS_FOR_UPDATE
REFERENCING OLD OLD_ROW
FOR EACH ROW
    DELETE FROM FLIGHT_DB2.CUSTOMER_ORDERS
    WHERE ORDER_NUMBER =
        (SELECT ORDER_NUMBER
         FROM FLIGHT_DB2.CUSTOMER_ORDERS T1
         WHERE OLD_ROW.ORDER_NUMBER = T1.ORDER_NUMBER);
LABEL ON TRIGGER CANCEL_CUSTOMER_ORDER
IS 'Cancel Customer Order';
```

When the SQL procedure **DELETE_CUSTOMER_ORDER** issues a delete operation against the view **CUSTOMER_ORDERS_FOR_UPDATE**, the delete operation is passed to the **INSTEAD OF** trigger, which deletes the order using the **CUSTOMER_ORDERS** base table. More examples of **INSTEAD OF** triggers are provided with the Data-Centric Development workshop.

9.4 Phase 3 and beyond

This section provides additional information regarding the process of maintaining and altering the existing database before the Flight/400 database reengineering team goes forward with the database modernization process. One of the main goals of the Flight/400 modernization effort is to reduce the time that is required to implement new business opportunities. To achieve this goal, it is important to continue to follow the modernization strategy when modifying a database object.

Forward engineering implies that someone has decided which files should be enhanced going forward. There is definite value in modernizing more files based on business priorities. This is an iterative process and might continue for an extended period.

Forward engineering definitely should not take the place of proper database modernization, but DDS defined files can be processed by using SQL statements and can be freely mixed with SQL defined tables in the same query. For example, there are no functional issues with queries that join physical files and SQL tables. So, in limited cases, it is a useful option to use SQL to access DDS files.

Figure 9-87 provides a high-level overview of the steps that must be taken before altering an existing table.

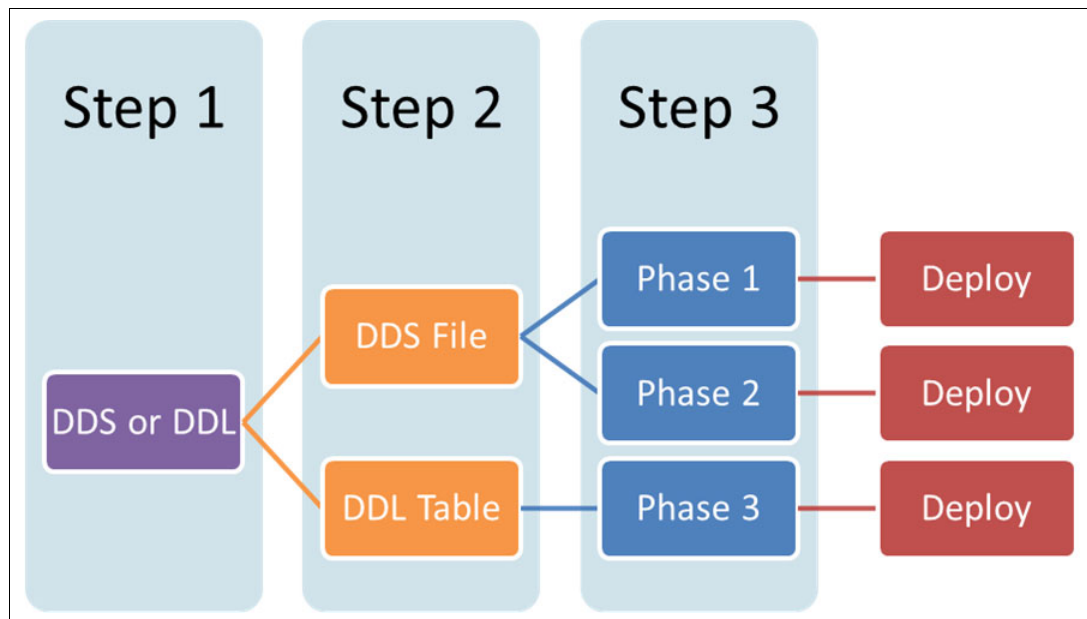


Figure 9-87 Steps for maintaining the database

The following list describes the process, which is shown in Figure 9-87, in more detail:

1. Determine whether the object is DDS or DDL defined.
2. Perform the following actions based on the step 1 findings:
 - a. For DDS, follow the processes that are defined in Phase 1 and Phase 2 before proceeding to step 3 on page 442.
 - b. For DDL, proceed to step 3 on page 442.

3. Use data modeling tools for forward engineering. Follow the recommendations that are provided with your database modeling tool of choice for managing and deploying database objects. The Flight/400 database team used InfoSphere Data Architect to take advantage of the logical data modeling capabilities.
4. Deploy the modifications after each phase. For minor changes to an existing DDS file, deployment can be done after Phase 2.

9.4.1 Managing database objects

It is the responsibility of the Database Engineer to maintain and deploy changes to the database. The database engineering team for the Flight/400 project was created before Phase 1. There are two approaches to maintaining and altering the existing database. Both include the usage of a data modeling tool, but differ in the starting point for making the changes. Both options are consistent with preferred practices. Here are the two options:

- ▶ Changing the DB2 for i database
- ▶ Changing the InfoSphere Data Architect PDM

Both options require a “check-out” procedure to create a copy of the database and a copy of the InfoSphere Data Architect physical data model. The following section describes that process.

When the physical database is created, use the reverse engineering feature to create a PDM representing the DB2 for i schema. This is done by creating a Data Design project with the same name as the business component and then dragging the DB2 for i for schema from the Data Source Explorer to the PDM, as shown in Figure 9-88.

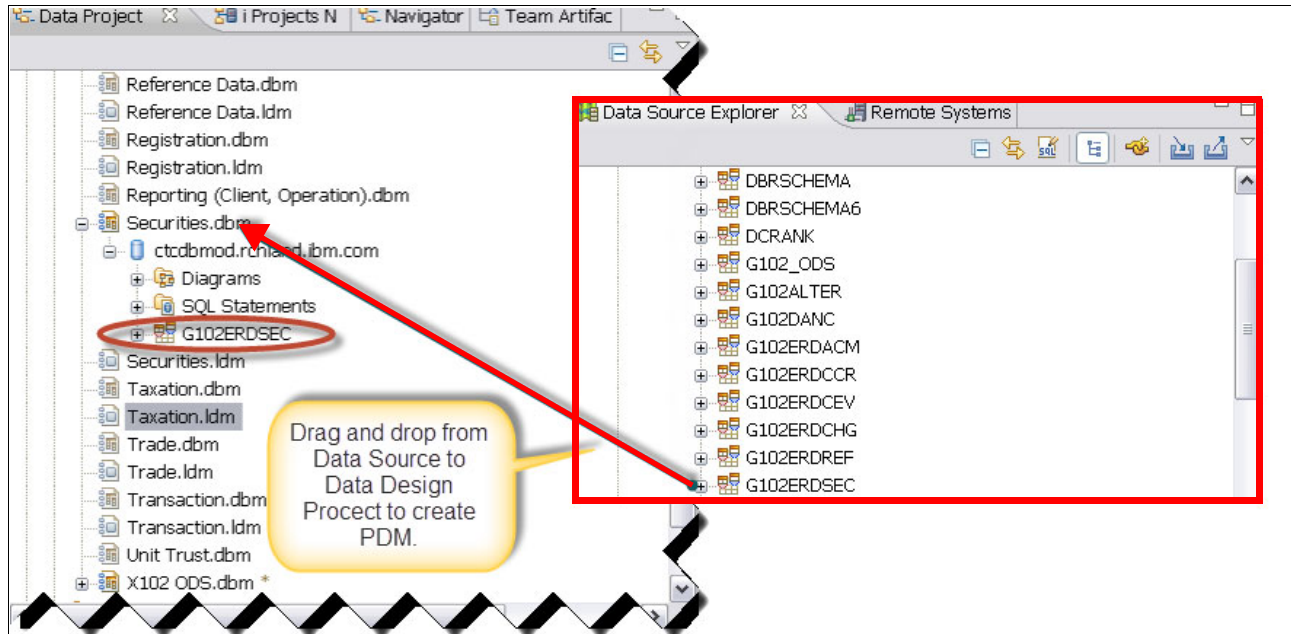


Figure 9-88 Physical data model from data source explorer

The InfoSphere Data Architect PDM is referred to as the Test Copy and the DB2 for i schema is referred to as the Original Source. You now have a test schema and model to implement further changes.

9.4.2 Changing the DB2 for i database

Whether re-engineering an existing database or developing a database from scratch, it is a preferred practice that modern, graphical data modeling tools be employed before, during, and after the development effort. Like a well-built house, a database should be designed to last forever and easily accommodate change. Just like any major construction project, there are blueprints, standards, and rules that must be followed to achieve success.

Preferred practices for defining new databases with SQL DDL

Here are the preferred practices when using SQL to define a DB2 for i database:

1. Never assume that SQL returns data in the order of an index or arrival sequence.

The preferred practice for both SQL and traditional applications is to explicitly designate the order in which you must access the non-unique rows. This can be accomplished by adding a row create time stamp column (only updated on **INSERT** by using **CURRENT TIMESTAMP** as a default), a DB2 maintained row change time stamp column, or a sequence column to the table and using this column, in combination with the ascending or descending designators to return the data in the required order.

For example, the SQL statement **SELECT... FROM T1 WHERE DupeKey1 = 1 ORDER BY DupeKey1, Create_Timestamp** returns all of the matching rows where DupeKey1 is equal to 1 and in the order they were added to table, regardless of RRN value. Specifying **DESC** for the **Create_Timestamp** column returns the same set of rows in descending sequence by the date they were added to the table; the last row that is added is the first row in the result set.

The row change time stamp can be used for update situations. For example, the SQL statement **SELECT... FROM T1 WHERE DupeKey1 = 1 ORDER BY DupeKey1, ROW CHANGE TIMESTAMP FOR T1** returns only the rows equal to DupeKey1 with the oldest changed row first. In addition, you can provide the **DESC** designator with the row change time stamp column to return the last changed row first in a list of duplicates.

For traditional I/O, use a keyed LF with the new column specified as the last key column with the appropriate ascending or descending designator.

2. Use DB2 for i auto-generation of column values capabilities.

Here is a link to the DB2 for i **CREATE TABLE** statement, which contains more information about auto-generated columns:

<http://publib.boulder.ibm.com/infocenter/series/v7r1m0/topic/db2/rbafzhctab1.htm>

3. Use constraints (UNIQUE, PRIMARY, FOREIGN, and CHECK).

This means that a new column representing the PK should be added to the existing tables. Make this column an **IDENTITY** column. This allows DB2 to generate the value that is a meaningless number. Use the **HIDDEN** attribute to hide this column and then use it to establish RI between related tables. If this is Phase 1 of a re-engineering effort, disable the foreign key constraint until the completion of Phase 2.

The following is a link to more information about constraints:

<http://publib.boulder.ibm.com/infocenter/series/v7r1m0/topic/db2/rbafzch1constraints.htm>

4. Follow established Relational Database Design techniques (also known as normalization).

Normalization is the process of efficiently and effectively storing data in a relational database. The goal is to eliminate redundant and duplicate data. This goal is accomplished by ensuring that each non-primary key fact is stored in exactly one place. In

addition, rules (constraints) are used to ensure that data dependencies and relationships are kept sound.

The design process follows a set of rules that are known as “normal forms”. There are basically five normal forms and the database designer follows these forms when creating the logical data model. When reverse-engineering existing databases, which are not well normalized, the order of the normalization process might deviate from the traditional practice.

5. Use integers for numeric data that does not require decimal precision.

An INTEGER can have a maximum positive value of 2 billion and use only 4 bytes of storage. A BIGINT is only 8 bytes; however, the maximum positive value is a billion times larger than an integer. Here is a link to DB2 for i limits:

<http://publib.boulder.ibm.com/infocenter/iseriess/v7r1m0/topic/db2/rbafzlimtabs.htm>

6. Use strong typing.

This means that if the data represents a value, such as date or time, then define the column as a date or time type.

7. Use appropriate numeric types.

Do not be stingy when defining the size of your numeric columns. Allow for growth.

Here are guidelines for defining numeric columns:

- Use NUMERIC if the value is limited to 0 - 9.
- Use INTEGER if decimal precision is not required.
- Use DECIMAL if decimal precision is required (that is, dollar amounts) or the value can be greater than +/-9,223,372,036,854,775,807.
- Use Floating point if you need large numbers, that is, greater than +/-9,223,372,036,854,775,807.

8. Define character fields appropriately.

Again, do not be too restrictive in size; however, avoid the use of CHAR for large text columns. A CHAR column is padded with blanks forcing a scan of the entire column during text searches. VARCHAR columns are NULL delimited. Searches stop when the NULL is encountered. Here are the preferred practices for defining text columns:

- Use CHAR for text fields under 20 characters in length or if the text is always equal to the column size.
- Use VARCHAR if the text field is greater than 20 characters in length.

In addition, the **ALLOCATE** keyword can be used to designate a fixed length for VARCHAR columns. This allocation provides both the performance benefit of null delimited text searches and the elimination of an extra I/O to the variable data storage area.

To use the **ALLOCATE** keyword, determine the maximum data size of the existing data in the column. Here is an example query (VARCHAR is required for CHAR columns):

```
SELECT LENGTH(VARCHAR((text_Col)) text_sizes, count(*) how_many
FROM T1 group by length(table_name) ORDER BY text_sizes DESC
```

Determine the 95th percentile of all column lengths and use this value for **ALLOCATE**.

9.4.3 Creating a table

As a preferred practice, create a template to ensure that all new tables contain the core fundamental items that are required by the business. When the template table is created, it can be “cloned” by using an SQL script or a graphical interface.

The new table template

Example 9-19 is an example of an SQL script, which can be used as a template for creating tables.

Example 9-19 Creating a table template SQL script

```
CREATE TABLE New_Table_Template
/* The FOR SYSTEM NAME clause is required if existing DDS LFs need to be
re-created over the new table. A meaningful short name is recommended even if DDS
is not involved in the reengineering effort*/
  FOR SYSTEM NAME short_name
(
  /* The position of the primary key (identity column) in the row is irrelevant
*/
  New_Table_ID FOR COLUMN NEWTABLEID BIGINT
    GENERATED BY DEFAULT AS IDENTITY IMPLICITLY HIDDEN
  /* Primary key constraint is defined here */
  PRIMARY KEY,
  /* Every table must have a unique key and it cannot be hidden */
  Application_Key FOR COLUMN APPKEY INTEGER UNIQUE,
  /* Optional column containing time stamp when row created. Can be used to order
in arrival sequence, regardless of row location in table */
  CREATE_ROW_TS FOR COLUMN WHEN_ADDED TIMESTAMP NOT NULL
    DEFAULT CURRENT_TIMESTAMP IMPLICITLY HIDDEN,
  ROW_CHANGE_TS FOR COLUMN WHEN_CHANGED TIMESTAMP
    GENERATED ALWAYS FOR EACH ROW ON UPDATE
    AS ROW CHANGE TIMESTAMP NOT NULL IMPLICITLY HIDDEN,
  CONSTRAINT New_Table_PK PRIMARY KEY( New_Table_ID ) )
/* Record format name is optional and not required by SQL. Might be useful when
using some i5/OS commands */
RCDFMT NEWTABLER;

LABEL ON COLUMN New_Table_Template
( New_Table_ID TEXT IS 'Identity Column for this table',
  Application_Key TEXT IS 'Unique key used by application',
  CREATE_ROW_TS TEXT IS 'Row Creation Timestamp' ,
  ROW_CHANGE_TS TEXT IS 'Row Change Timestamp')
```

Using LIKE to create a table from the template

Example 9-20 is an example of an SQL script that can be used to create a table that is based on the previous template (Example 9-19) for new tables.

Example 9-20 Creating a table that is based on the template in Example 9-19

```
CREATE TABLE New_Table_Name
/* The FOR SYSTEM NAME clause is required if existing DDS LFs need to be
re-created over the new table. A meaningful short name is recommended even if DDS
is not involved in the reengineering effort*/
  FOR SYSTEM NAME short_name
```

```

LIKE New_Table_Template
/*The INCLUDING clauses ensure that the appropriate attributes and defaults are
included in the new table*/
INCLUDING IDENTITY COLUMN ATTRIBUTES
INCLUDING COLUMN DEFAULTS
INCLUDING IMPLICITLY HIDDEN COLUMN ATTRIBUTES
INCLUDING ROW CHANGE TIMESTAMP COLUMN ATTRIBUTES;

/* Always commit your work */
COMMIT;

```

After running the SQL script in Example 9-20 on page 445, the new table is now ready to be modified by using the System i Navigator Table Definition properties window.

Using the Table COPY function from System i Navigator

The System i Navigator client tool provides the means to create and maintain database objects. The Database view contains the various tools that can be used to run SQL scripts and graphically display and alter existing database objects.

The Table Copy option can be used as an alternative to the SQL **CREATE TABLE ... LIKE** statement. To use this feature, expand **Databases** → **Schemas**. Find and expand the schema that contains the **NEW_TABLE_TEMPLATE**. Expand **Tables** and right-click the **NEW_TABLE_TEMPLATE** object. Select **Copy**. Right-click the schema name and select **Paste Definition**. Refresh the table list and find the new copied table (it is quoted and contains “Definition” in the Table name).

This process is shown in Figure 9-89.

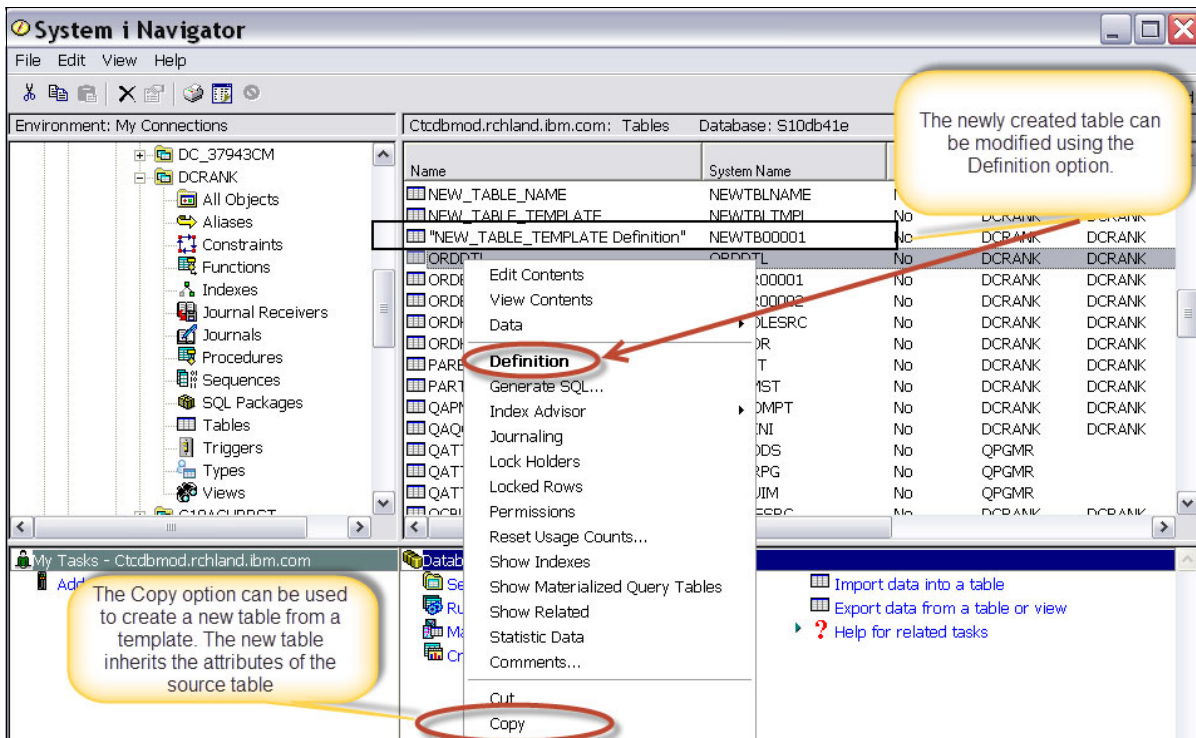


Figure 9-89 System i Navigator table copy function

Right-click the copied table and select **Rename**. Enter the new table long and short names and then click **OK** or press Enter, as shown in Figure 9-90.



Figure 9-90 System i Navigator table Rename window

9.4.4 Modifying a table by using the graphical interface of IBM i Navigator

If your business does not have a development team or you are simply in maintenance mode, then System i Navigator might be all that you need to modify a table. You can use the System i Navigator Table Definition window to add, change, or delete columns from an existing table. The Definition window can also be used to add, change, or delete constraints.

The Definition window

Right-click an existing table (this can be a new table that is created from a template or an existing table requiring modification) and select **Definition**. The Definition window opens and shows the Table tab. If this is a new table, you can add the table description in the Text field, as shown in Figure 9-91.



Figure 9-91 System i Navigator table Definition window

Adding columns and constraints

Click the **Columns** tab to open a grid of the existing columns in the table. If this is a new table, verify that the column names and attributes are copied successfully. Scroll to the right to see and verify additional attributes (identity, row change time stamp, and so on). When you are satisfied, click **Add** to add more columns to the table, as shown in Figure 9-92.

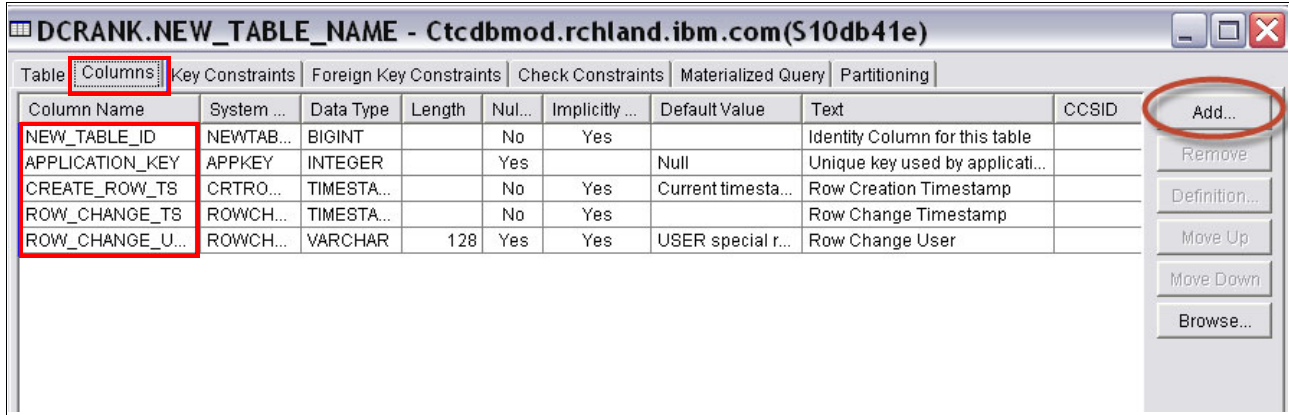


Figure 9-92 System i Navigator Table Definition Columns tab

After you click **Add**, the New Column window opens. Enter the new column name, data type, text, null capability and click **Add**. The new column is added to the table definition. The New Column window remains open until all new columns are entered. When the last column is entered, click **Close** to return to the Column grid, as shown in Figure 9-93.

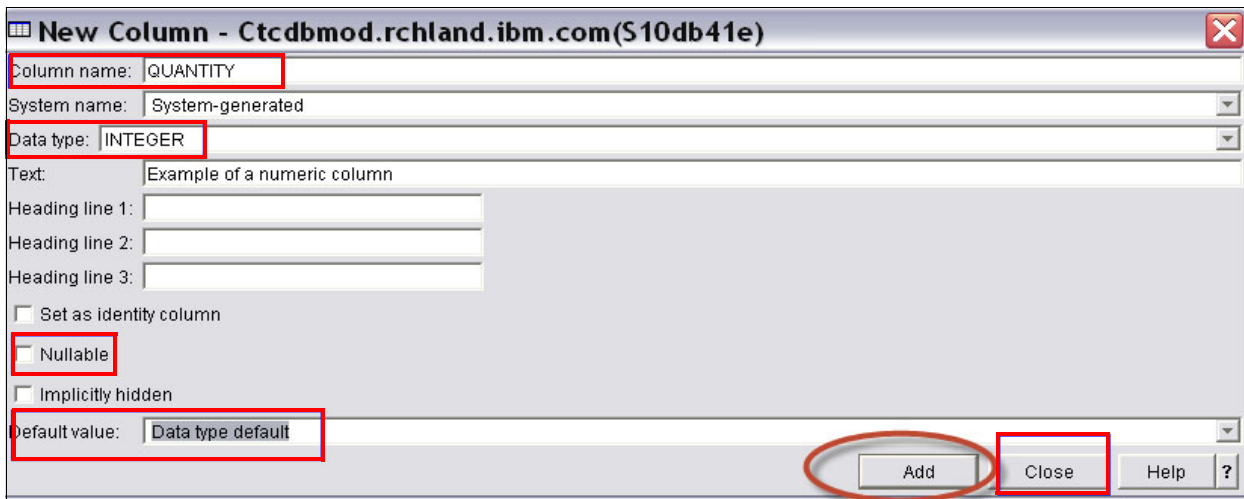


Figure 9-93 Table Definition Add New Column window

CREATE TABLE LIKE does not create the Primary and Unique constraints; however, the Copy function does. Constraints should be added (or verified) as part of the new table process. To begin, click the **Key Constraint** tab in the Definition window to open the New Key Constraint window. Define the primary key constraint by typing in a constraint name, selecting **Primary key**, and selecting the Identity column as the primary key column. Press **OK** to add the constraint to the table definition, as shown in Figure 9-94 on page 449.

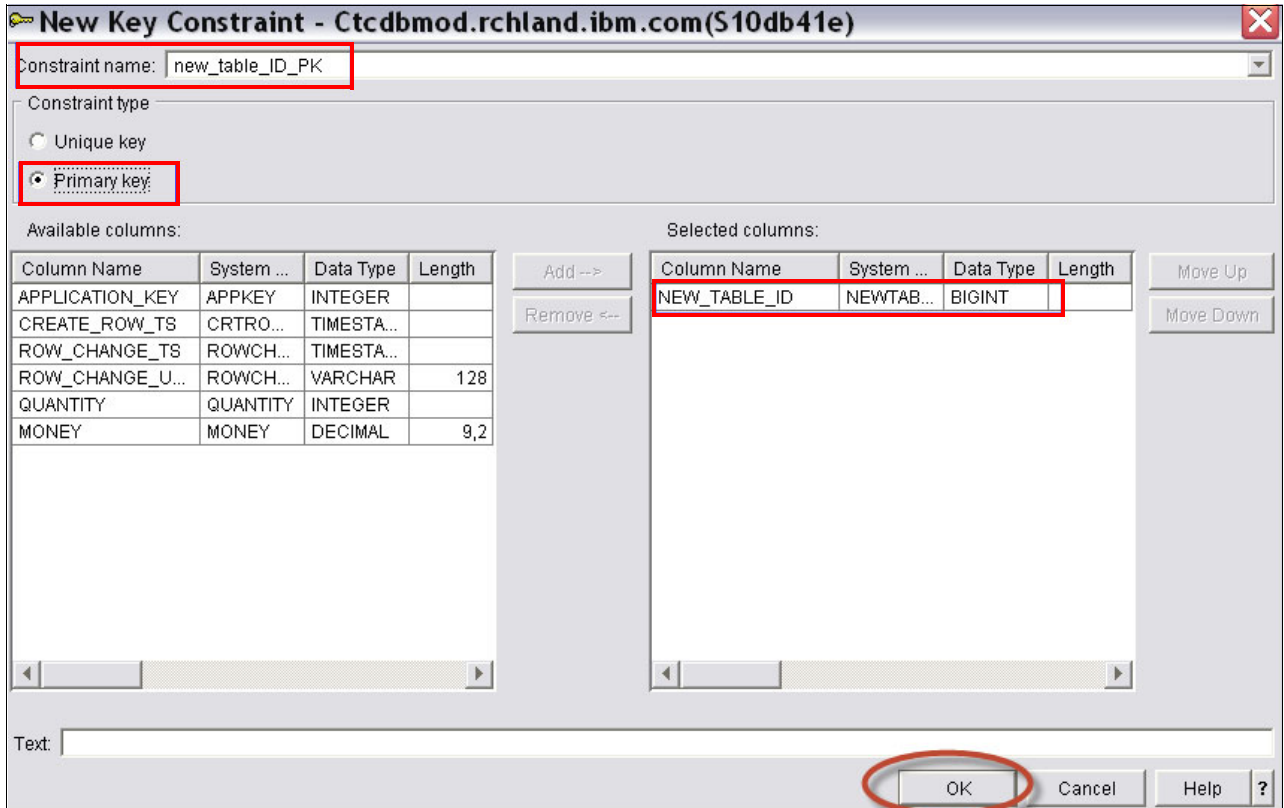


Figure 9-94 Table Definition Add Primary Key Constraint window

Repeat the previous process and create a unique key constraint for the application key column. When completed, there should be two constraints that are defined for this table, as shown in Figure 9-95.

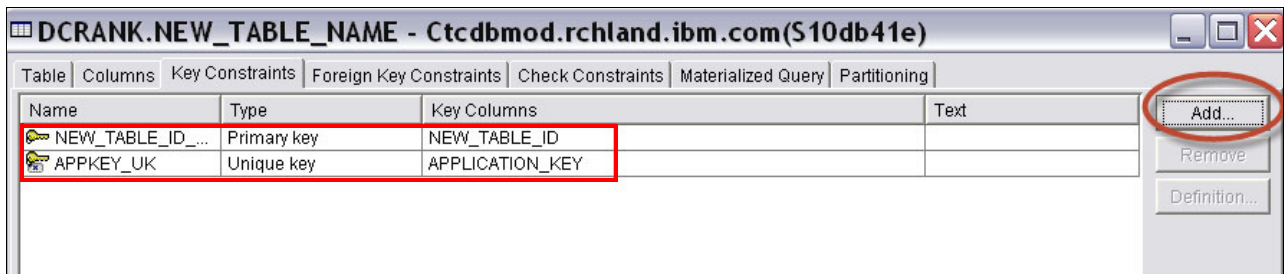


Figure 9-95 Table Definition Key Constraints List window

Modifying an existing column

To modify an existing column, open the Definition window and click the **Column** tab to display the column grid. Find the column to be changed and click **Definition**. This process is shown in Figure 9-96.

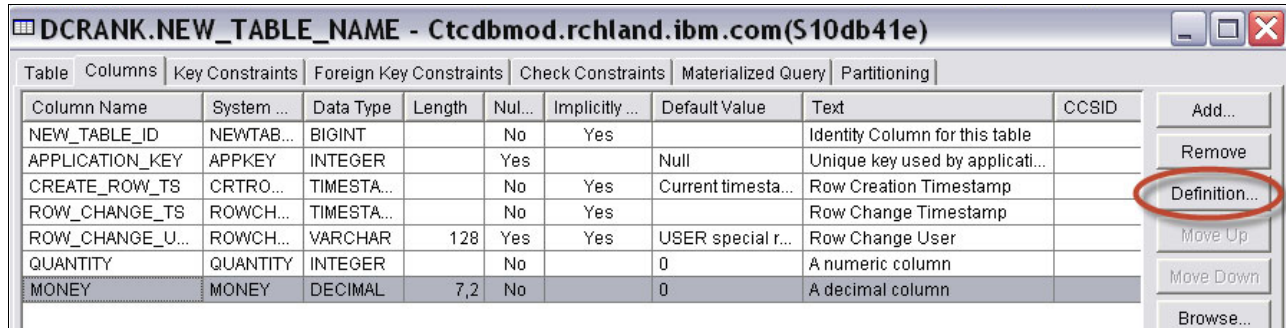


Figure 9-96 Selection a column for modification

The Column Definition window opens. Make your changes and click **OK** to return to the column grid, as shown in Figure 9-97.



Figure 9-97 Modifying a column

Capturing the SQL ALTER TABLE statement

The final step, before deploying the modified table, is to capture the **ALTER TABLE** statements that are generated by System i Navigator. To do so, click **Show SQL** in the last window that is used to make changes. Figure 9-98 on page 451 shows the Column grid window with the **Show SQL** button active after modifying an existing column.

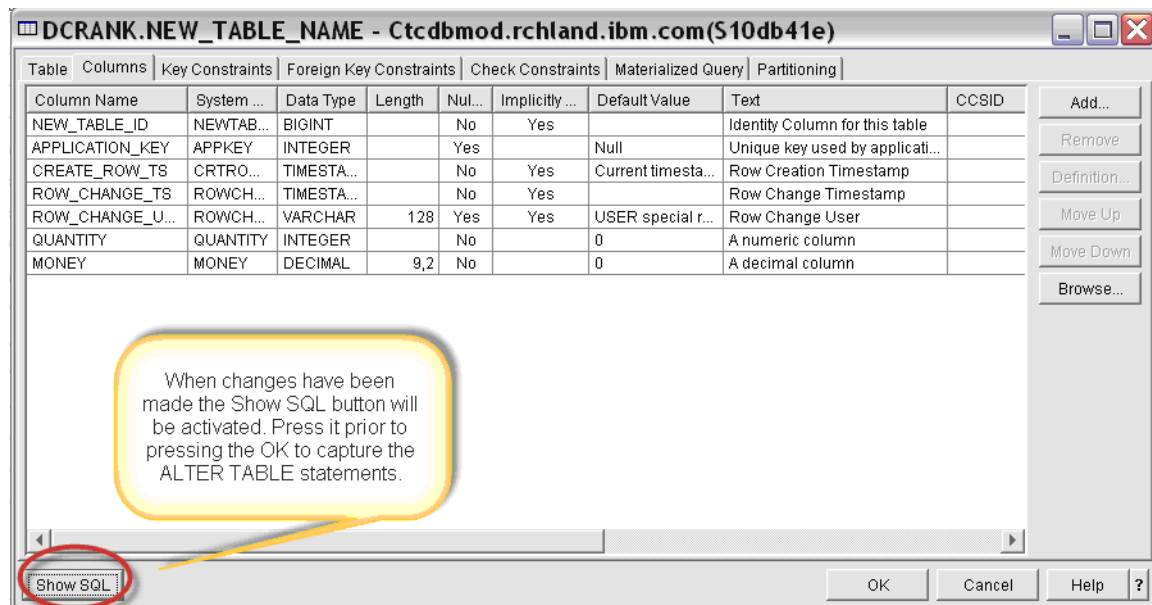


Figure 9-98 Showing the SQL based on the changes to the definition

Figure 9-99 is an example of the SQL statements that are generated after adding new columns and constraints to a new table.

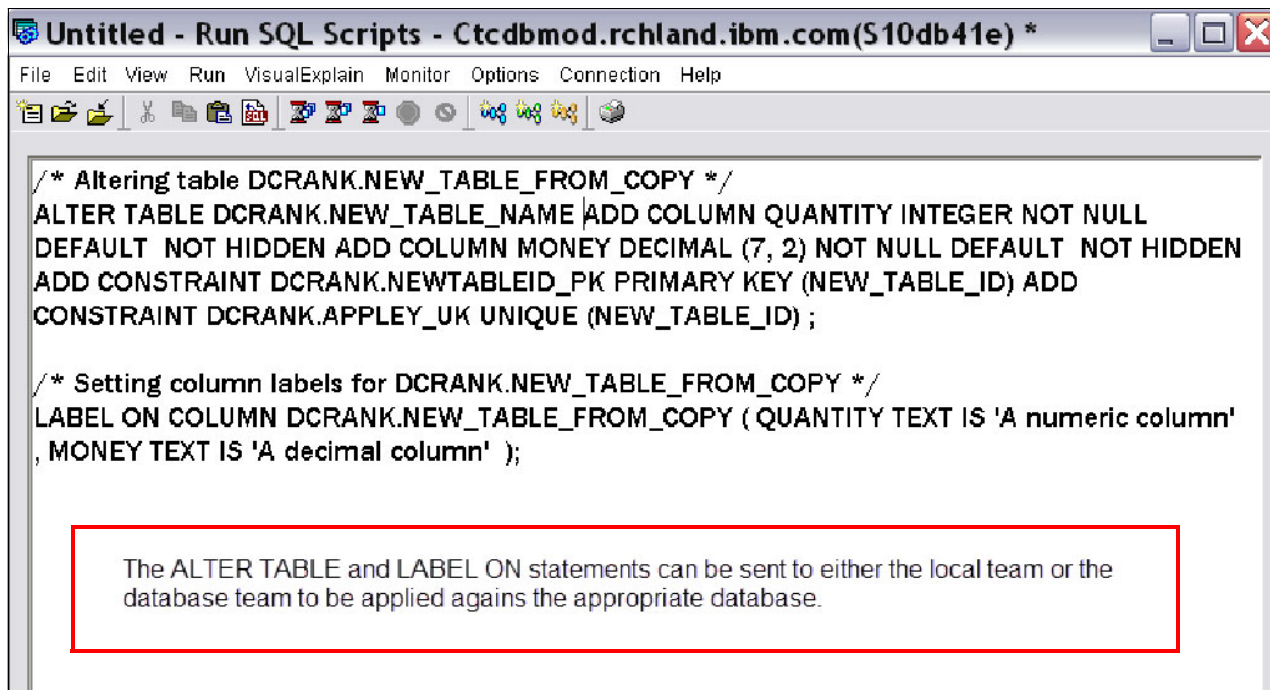


Figure 9-99 Running an SQL script with ALTER constraints statements from Show SQL

Figure 9-100 is an example of the SQL statements that are generated after modifying an existing column.



Figure 9-100 Running an SQL script with ALTER column statements from Show SQL

These **ALTER** scripts should be retained in change control because they can be used to implement the database changes on other instances.

Synchronizing the PDM

When you are using a data modeling tool, it is important that the model source is kept in sync with the instantiated database. Both the IBM Data Studio and InfoSphere Data Architect tools can compare the PDM copy to the original DB2 for i source.

Comparing the PDM to the DB2 for i schema

From within the Data Design project, open and expand the PDM for the Test copy. Right-click and select **Compare With** → **Original Source**, as shown in Figure 9-101.

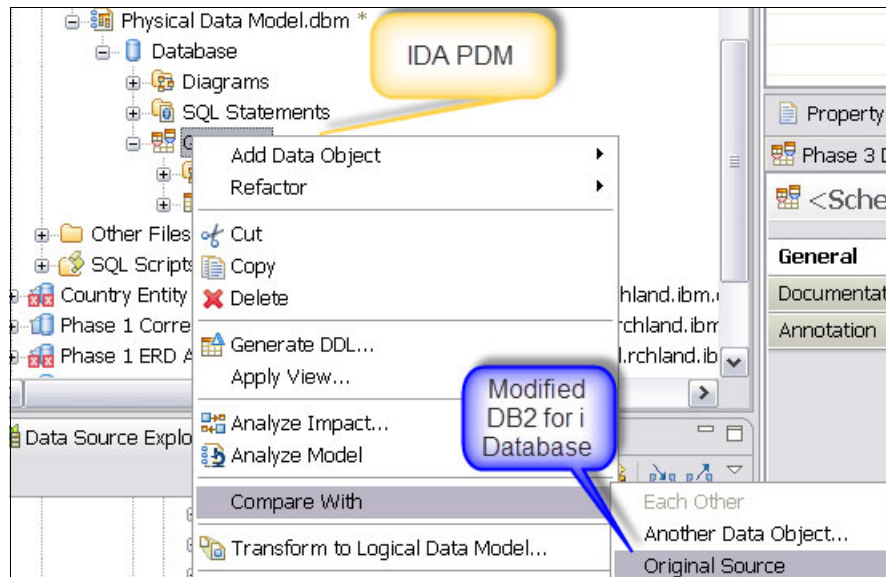


Figure 9-101 Right-clicking a schema name within the Data Design Project

Synchronizing the PDM from the DB2 for i schema

The Compare window opens. The upper part of the window contains the Test copy database objects on the left and the Original DB2 for i source database objects on the right. Only objects with differences are shown, as in Figure 9-102 on page 453.

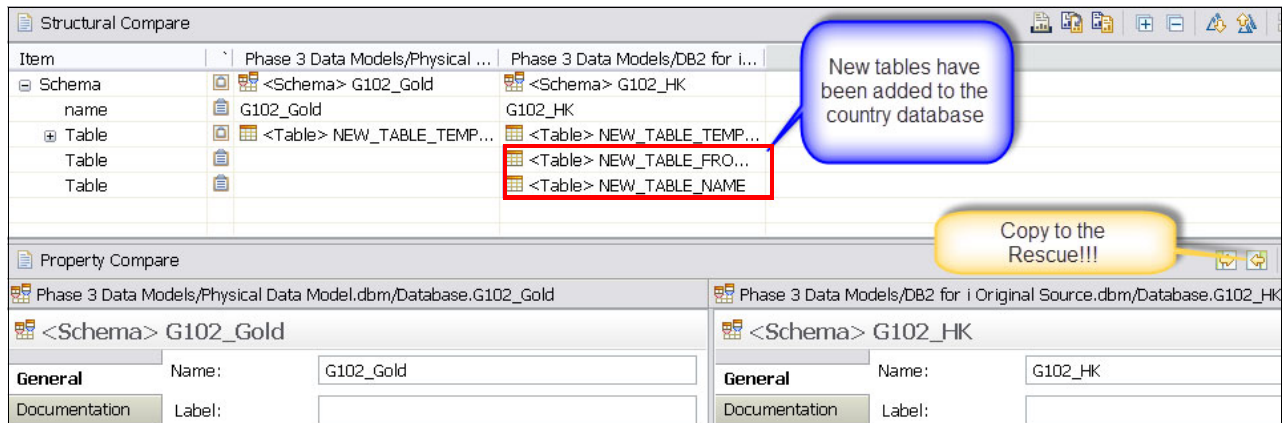


Figure 9-102 Structural Compare window after choosing Compare With

After ensuring that the differences are valid, click **Copy** from the icon that is in the upper-right corner of the Property Compare window. After using this option, the Test copy is synchronized with the Original DB2 for i source. The Compare window automatically is refreshed with the changes. Objects that are changed successfully have a check mark in the box directly to the left of the object, as shown in Figure 9-103.

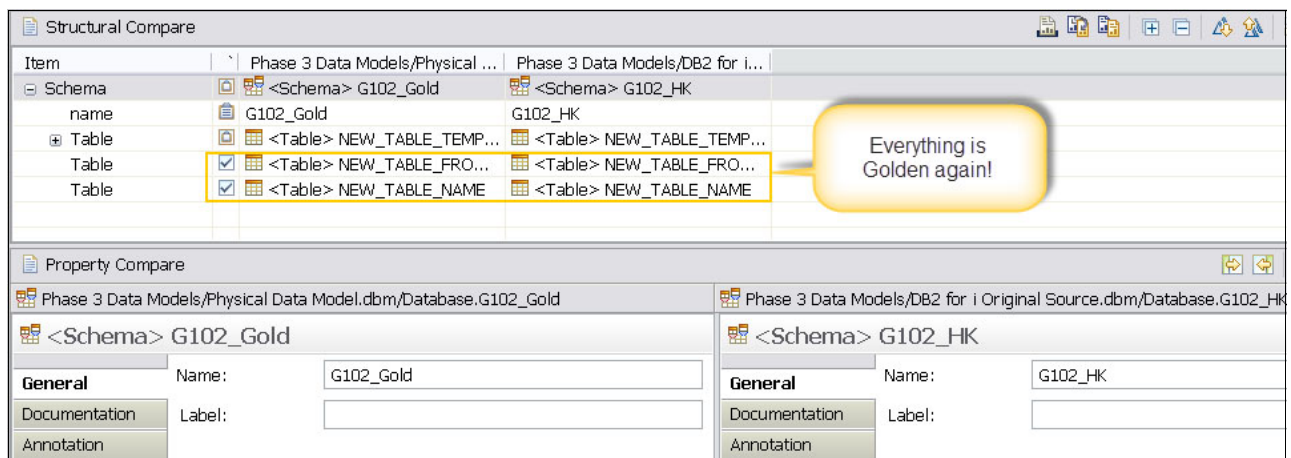


Figure 9-103 Structural Compare window after using the Copy to Right option

9.4.5 Changing the PDM

The option of starting with changes to the PDM is the correct choice when making changes to the Test copy.

Creating a table from a template

The IBM Data Studio and InfoSphere Data Architect tools are the preferred tools for data-centric development. These tools contain the same capabilities as System i Navigator for maintaining databases with several more ease-of-use features. One of those features is the ability to create a template for commonly used SQL DDL statements, such as **CREATE TABLE**.

We used the following process to create our table template in our example:

1. Creating a diagram
2. Using the palette to add objects

3. Using the Properties view to add columns
4. Modifying column attributes
5. Adding constraints
6. Adding a sequence object
7. Validating the template

Creating a diagram

To create a template for the creation of related tables, start by creating a diagram in the Future State physical data model. We named the diagram Templates in our example.

Using the palette to add objects

Find the Palette view and expand the **Data** folder. Drag the Table icon on to the blank diagram. Repeat this task two more times. Click each new table object, open the General tab in the Properties view, and rename each object, as shown in the Table 9-8.

Table 9-8 Table update values

Default name	New long name	Short name	Description
Table1	Parent_Long_Name	PARENT	Identifying Parent: Responsible for the existence of child
Table2	Child_Long_Name	CHILD	Dependent table: Cannot exist without at least one parent
Table3	Significant_Other	SPOUSE	

Figure 9-104 shows the updates from the table.

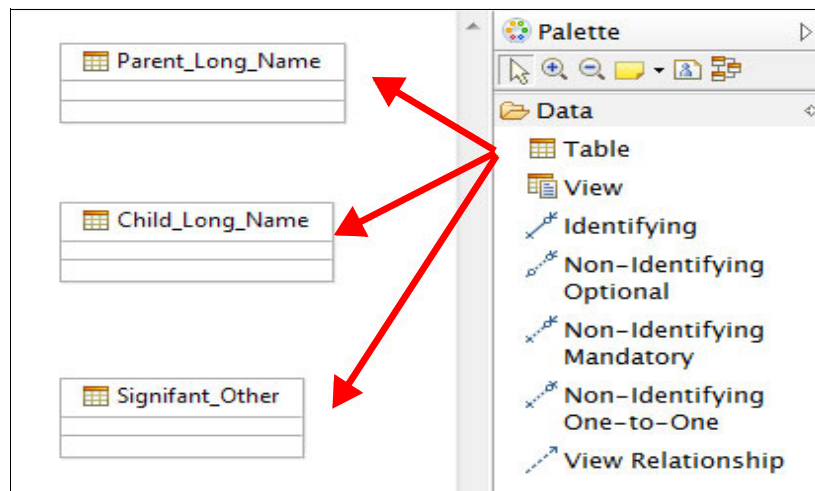


Figure 9-104 Table diagram with updates

Using the Properties view to add columns

Click each table and, using the Columns tab in the properties view, define the core fundamental items for each one. The child table does not have a primary key column. The diagram now appears as shown in Figure 9-105 on page 455.

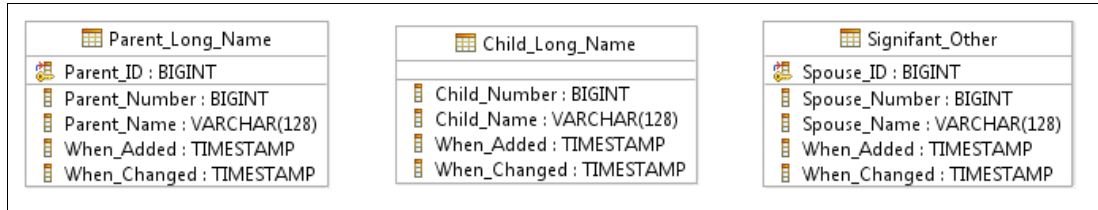


Figure 9-105 Diagram with tables and columns

Modifying column attributes

Click each column and select **Type** from the Properties view. Modify the attributes in each table as follows:

1. For each primary key, modify the Expression to As Identity: we select the **Generated** option, specified BY_DEFAULT as the Generate type, and select the **Hidden** attribute at the bottom of the view.
2. For each VARCHAR column, we enter 50 in the Allocate tab.
3. For each When_Added column, we specify CURRENT_TIMESTAMP as the default (NOT NULL) and select the **Hidden** attribute.
4. For each When_Changed column, we check the **Row Change Timestamp** and **Hidden** check boxes.

Adding constraints

Returned to the diagram and select the Identifying tool from the Palette to create an owner-ship referential constraint between the Parent table and the Child table. Use the Non-identifying Mandatory tool to create a referential constraint between the Spouse table and the Child table.

The primary key columns were automatically added as foreign keys to the Child table. For each foreign key, we select the **Hidden** attribute because we want our primary keys to be meaningless (auto-generated as Identity columns) and not exposed by `SELECT *` to the Child table.

We ensure that the RI rules are CASCADE and RESTRICT for the Parent and Spouse constraints. The business rules dictate that if the Parent is deleted, then the child is deleted (CASCADE). Prevent the spouse from being deleted if the child exists (RESTRICT). The child can be deleted at any time.

The diagram now looks like Figure 9-106.

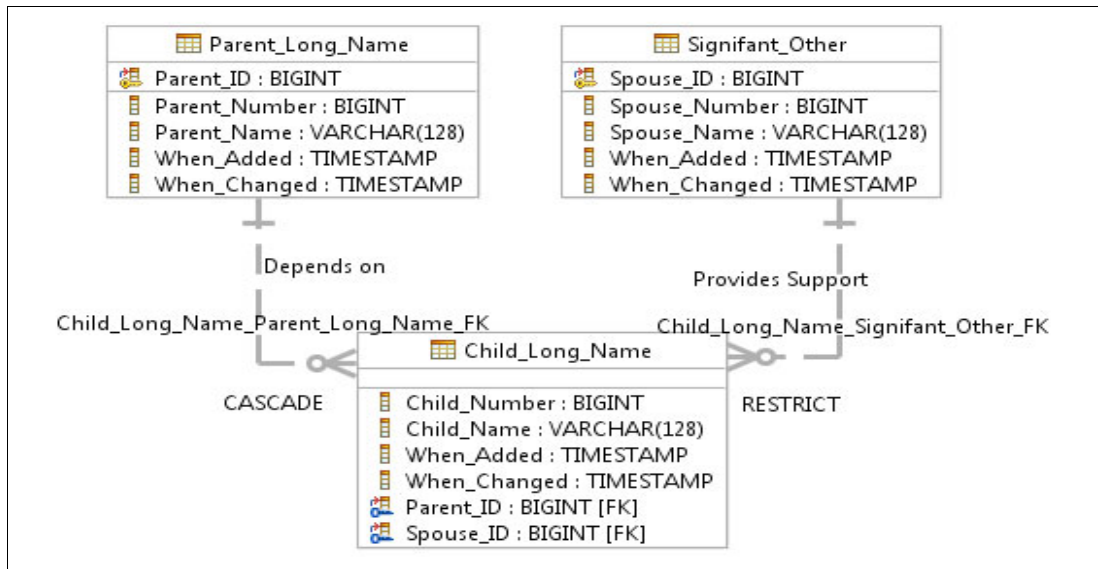


Figure 9-106 Diagram view with constraints added

The diagramming tool does not have an option for adding UNQUE constraints. To add these constraints, return to the Data Project Explorer view. Right-click each column ending in _Number and chose Add Data Object Unique Constraint. A new UNIQUE constraint is added to each table, as shown in Figure 9-107.

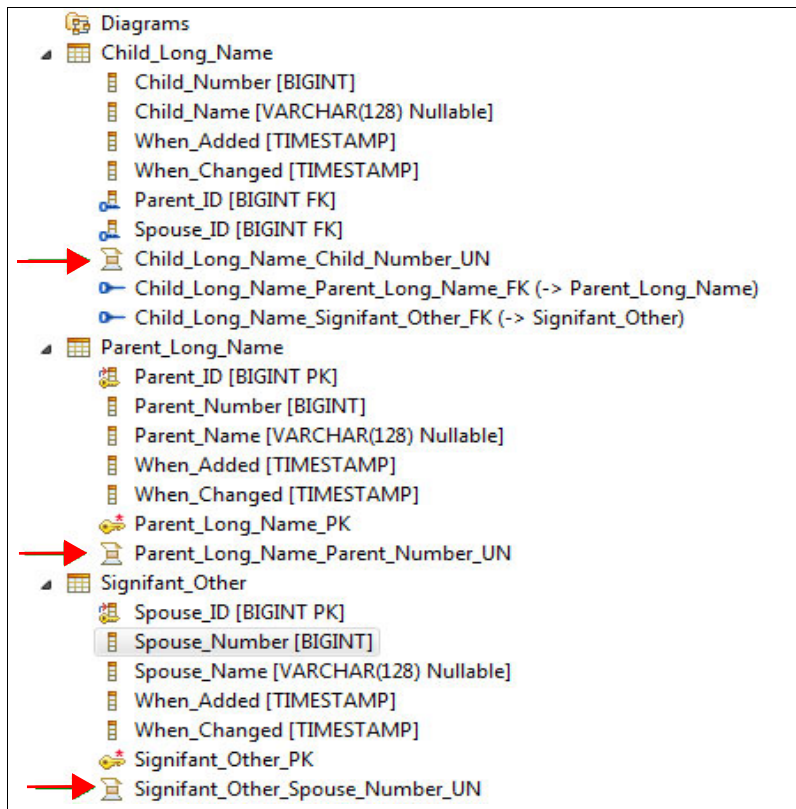


Figure 9-107 Unique constraints added to each table

Adding a sequence object

To complete the template, right-click the schema name (we chose QGPL because we did not intend to deploy the template) in the Data Project Explorer and select **Add Data Object** → **Sequence**. We named the Sequence Next_Unique_Number. This sequence is used to generate the next key value for the columns ending with _Number when a new row is added to the table.

Validating the template

To validate that the DDL generated for the template works, right-click the schema name (QGPL) in the Data Project Explorer and select **Generate DDL**. We look at the DDL in the Generate DDL window and verify the following items:

- ▶ The **CREATE SEQUENCE** object statement is there.
- ▶ The primary and foreign key columns are hidden.
- ▶ Identity and row change time stamp columns are generated as requested.

After verifying the DDL, click **Cancel** to leave this window.

Figure 9-108 shows the verification window.

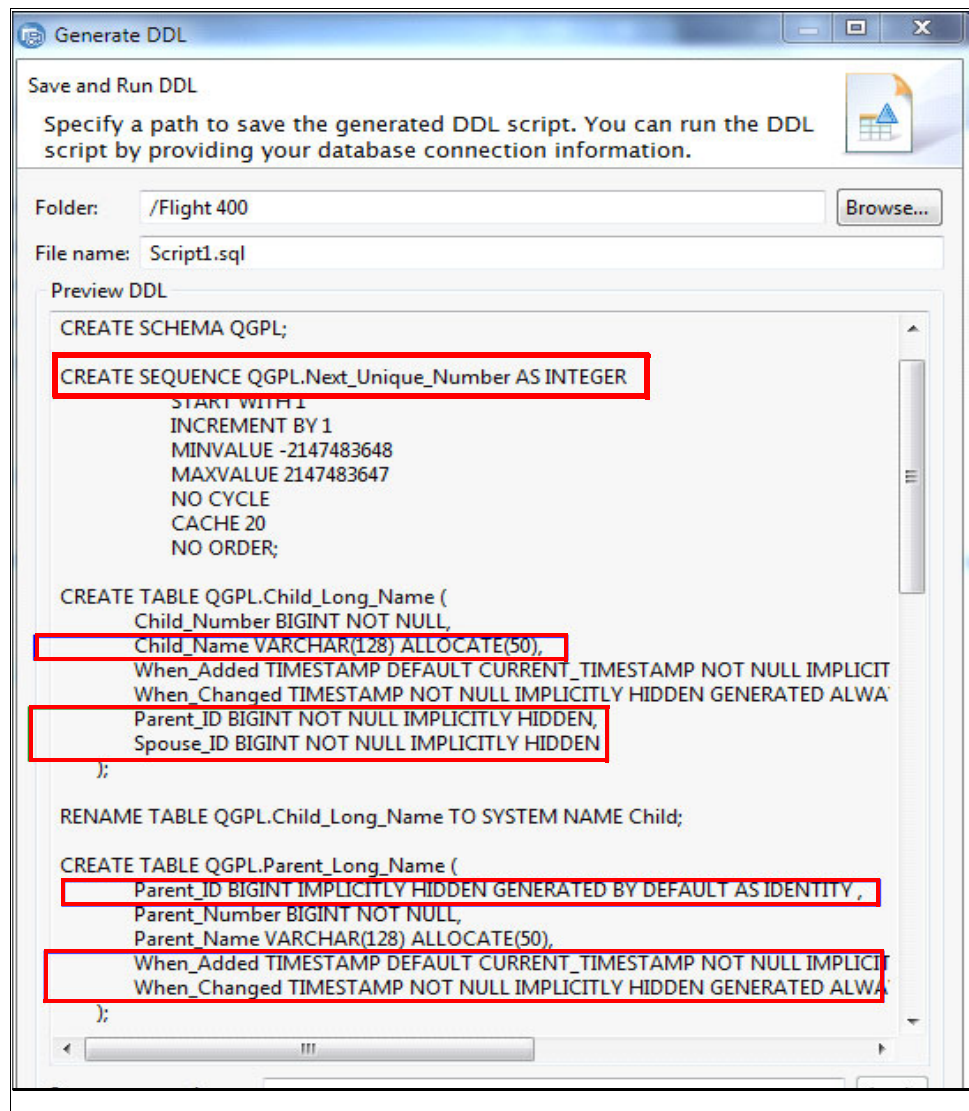


Figure 9-108 Verifying the Generate DDL window

Using the Data Model templates

In our example, we copy the template diagram as the first step when we create an application, which forces us to think in terms of relationships and sets. When refactoring an existing table, you can copy the columns from the template tables and copy them to the table that is being modified.

Remember, a table without a relationship is a *flat* file.

Using the supplied DDL templates

In addition to data modeling diagrams, IBM Data Studio and InfoSphere Data Architect provide templates that are based on standard SQL CREATE DDL statements. To create a table that is based on the CREATE TABLE template, expand the Data Design project, right-click the **SQL Scripts** folder, and select **New** → **SQL or XQuery Script**, as shown in Figure 9-109.

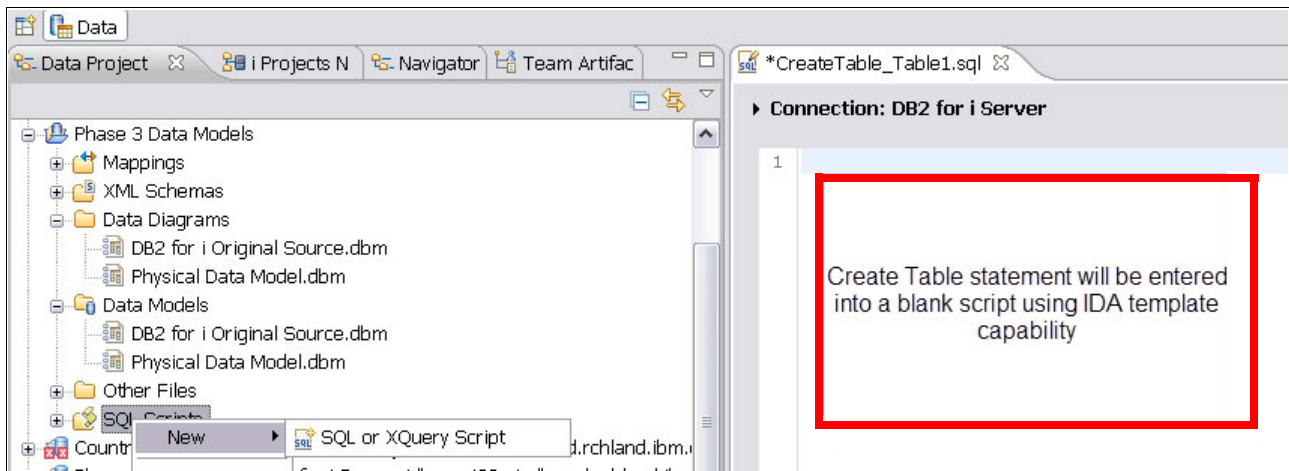


Figure 9-109 Creating an SQL script

From within the SQL script editor, type the characters 'cr' and then press Ctrl+Space to open a list of SQL DDL statement templates that begin with CR, as shown in Figure 9-110.

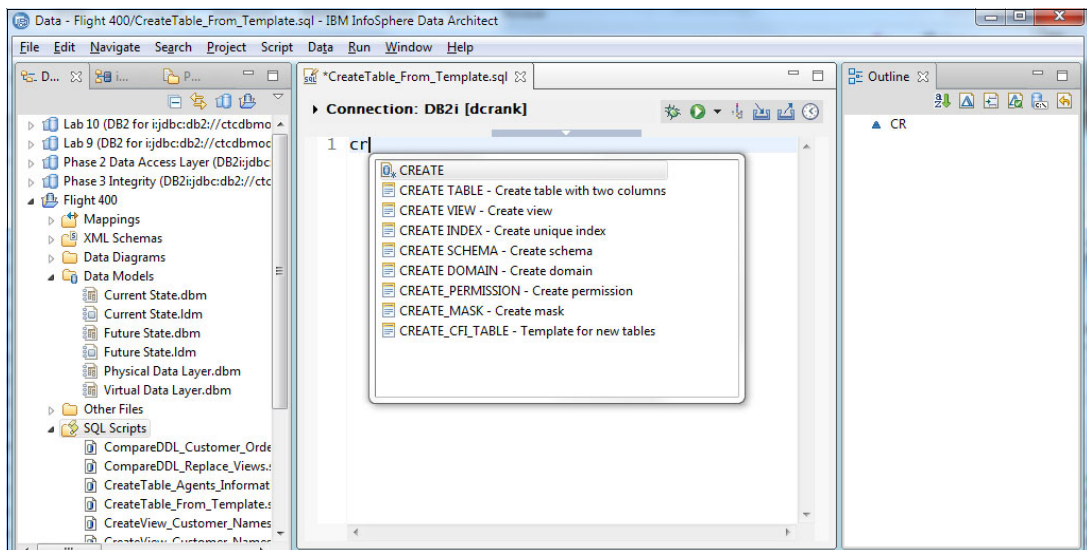


Figure 9-110 DDL templates starting at CR

Double-click the template that is named CREATE TABLE to populate the SQL editor window with the CREATE TABLE template. This template appears to be identical to the template that is used for System i Navigator. The main difference is that substitution variables are defined for the parts of the statement that must be customized. These parameters are highlighted or boxed, as shown in Figure 9-111.

```
CREATE TABLE table_name (
  short table name ID FOR COLUMN short name ID BIGINT GENERATED BY DEFAULT AS IDENTITY (
    MINVALUE Start ID Range MAXVALUE End ID Range)
  IMPLICITLY HIDDEN ,
  Application Key FOR COLUMN APPKEY INTEGER UNIQUE,
  CREATE_ROW_TS FOR COLUMN CRTROWTS TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP IMPLICITLY HIDDEN ,
  ROW_CHANGE_TS FOR COLUMN ROWCHGTS TIMESTAMP GENERATED ALWAYS FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP NOT NULL IN
  ROW_CHANGE_USER FOR COLUMN ROWCHGUSER VARCHAR(128) ALLOCATE(18) CCSID 37 DEFAULT USER IMPLICITLY HIDDEN ,
  CONSTRAINT table_name_PK PRIMARY KEY( table_name_ID ) )
  RCFMT record format ;

LABEL ON COLUMN table_name
( table_name_ID TEXT IS 'Identity Column for this table',
  Application Key TEXT IS 'Unique key used by application',
  CREATE_ROW_TS TEXT IS 'Row Creation Timestamp' ,
  ROW_CHANGE_TS TEXT IS 'Row Change Timestamp' ,
  ROW_CHANGE_USER TEXT IS 'Row Change User' ) ;

RENAME TABLE table_name TO SYSTEM NAME short_table_name;
```

Keying into the highlighted code replaces that code, and all references, with the new code. Start with table_name, short_table_name, identity ranges, record format, etc.

Figure 9-111 SQL script with generated DDL statements from a template

Entering data into the highlighted box results in the data being copied in each variable with the same name. In the example, table_name is the first substitution variable and is repeated throughout the script. The next variable is short_table_name, and it is also repeated throughout the script. There are more variables that might or might not be repeated but must be changed. Add any additional columns to the script. After entering all variable information, the SQL script appears similar to what is shown in Figure 9-112.

```
*CreateTable_Table1.sql
Connection: DB2 for i Server

1 CREATE TABLE new_table_From_IDA (
2   newtbl1da1_ID FOR COLUMN ida1ID BIGINT GENERATED BY DEFAULT AS IDENTITY (
3     MINVALUE 1 MAXVALUE 2000000000000000000)
4   IMPLICITLY HIDDEN ,
5
6   quantity integer not null default 0,
7   money decimal(7,2) not null default 0,
8
9   Application Key FOR COLUMN APPKEY INTEGER UNIQUE,
10  CREATE_ROW_TS FOR COLUMN CRTROWTS TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP IMPLICITLY HIDDEN ,
11  ROW_CHANGE_TS FOR COLUMN ROWCHGTS TIMESTAMP GENERATED ALWAYS FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP NOT NULL IN
12  ROW_CHANGE_USER FOR COLUMN ROWCHGUSER VARCHAR(128) ALLOCATE(18) CCSID 37 DEFAULT USER IMPLICITLY HIDDEN ,
13  CONSTRAINT new_table_From_IDA_PK PRIMARY KEY( newtbl1da1_ID ) )
14  RCFMT newtbl1dar ;
15
16
17 LABEL ON COLUMN new_table_From_IDA
18 ( newtbl1da1_ID TEXT IS 'Identity Column for this table',
19   Application Key TEXT IS 'Unique key used by application',
20   CREATE_ROW_TS TEXT IS 'Row Creation Timestamp' ,
21   ROW_CHANGE_TS TEXT IS 'Row Change Timestamp' ,
22   ROW_CHANGE_USER TEXT IS 'Row Change User' ) ;
23
24 RENAME TABLE new_table_From_IDA TO SYSTEM NAME newtbl1da1;
```

Add additional columns here. Press F5 to deploy to G102 database when complete.

Figure 9-112 Adding more columns to the CREATE TABLE statement

When all of the required fields are complete, press F5 to deploy the new table to the DB2 for i database (typically a test database).

Modifying the table attributes

The IBM Data Studio and InfoSphere Data Architect tools can modify the attributes of existing database objects.

Using the Table Properties view to change the attributes of an existing column

Click the table that needs alteration and select **Columns** from the Table Properties view. Find the column that you want to change in the column grid. All attributes can be changed directly on the grid. Change the attributes that are required, as shown in Figure 9-113.

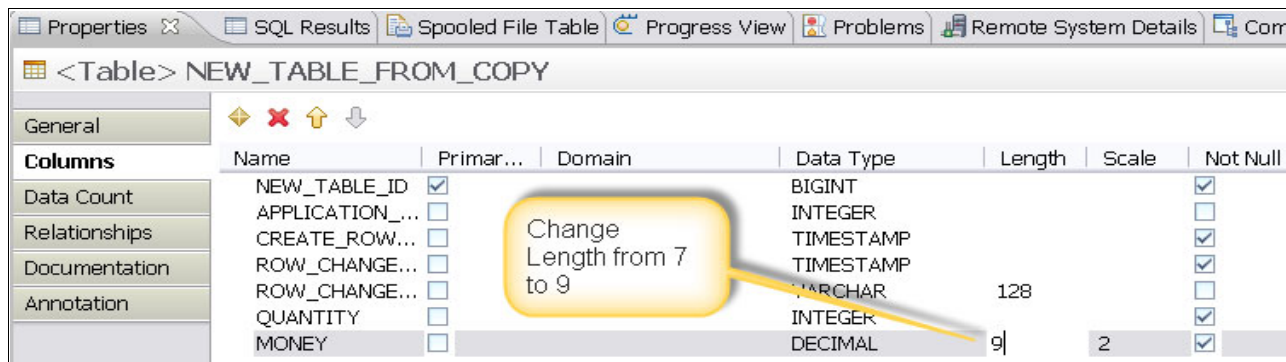


Figure 9-113 Updating the properties of an existing column

When all changes are made, save the changes to the open Table file.

9.4.6 Database update scenarios

Changes to the application requirements always drive more changes in the database. The beauty of database modernization is that modernized files can be changed fairly easily without requiring any changes to the existing application logic because they are no longer directly aware of the format of the underlying database files.

The following list provides examples of some typical database changes that become easier to implement when a file is modernized. Some can be implemented at the Phase 1 level using surrogate logical files, and some depend on using SQL to access the data in Phase 2.

- ▶ The DBE might decide that an existing column must be increased in size.
 - If the file has not been modernized, modernize it, including creating the surrogate logical files.
 - Increase the length of the column in the underlying SQL table to the new longer length using the process described in “Using the Table Properties view to change the attributes of an existing column”. The version of the table with the longer column value can be deployed on any instance of the database. Programs using the surrogate logical file can process any data that still fits in the field size that is specified in the logical file's DDS.
 - Programs that must access the longer version of the column can use either a new logical file or an SQL view. Programs that use SQL to access database files/tables are more compatible with later versions because they are less directly exposed to the physical layout of the data. Referencing the value directly in the table with Record Level Access breaks the rule of isolating program logic from the actual format of the underlying data.

- If the file is populated with values that are longer than the size in the surrogate LF, the behavior on reading the data depends on the data type. For character columns, programs using the surrogate logical file read the left part of the longer value. For numeric columns, programs get an overflow. Care must be taken to avoid deploying programs that put in longer values without ensuring that the other programs are impacted.

Note: Tools such as X-Analysis can also be used to determine the number of programs that are impacted by this change. If this change impacts only two programs, they can be revised. If it impacts 200 programs, the surrogate logical file approach can mask the change from the application programs.

- ▶ The DBE might decide to add a column to an existing table. This process is similar to the previous example of changing a column length.
 - If the file is not modernized, modernize it, including creating the surrogate logical files.
 - Add the column in the underlying SQL table by using the process that is described in “Adding columns and constraints” on page 448. The version of the table with the new column can be deployed on any instance of the database if it has a default value. This is required to ensure that the surrogate logical files are updatable.
 - Programs that do not need to reference the new column are unchanged because the new column is not included in their surrogate logical files.
 - Programs that must access the new column can use either a new logical file or an SQL view.

The DBE might decide that new function requires a new table for site information. Because the surrogate logical files present only the columns that are currently referenced by the existing application programs, a table can be altered to have a SITE_ID column that contains a foreign key that references one or more rows in the newly defined SITE_INFO table. (This can also be implemented by using the meaningless surrogate primary key fields that are added as part of database modernization.) The value can be populated using an SQL update statement and then used by new queries and applications with no impact to existing programs. This column can be also used in views in a consolidated database to limit visibility of the data.

A business might have a business requirement to eliminate a long running batch job that archives data by moving the records from a current file to an archive file. The file is unavailable when archival is in progress. Assume that the current table is named TRANS and the archive table is named ARCHTRANS and that data is moved when it is three years old. If the file is modernized, it has the time stamp when the row was created in the CREATE_ROW_TS column. That makes it possible to create views that select the archived data and the current data. For example, if the archival policy says that data is archived after three years, the views can look like what is shown in Example 9-21.

Example 9-21 Creating a view for 3-year archival

```

CREATE VIEW TRANS AS
  SELECT * FROM TRANS_T
  WHERE YEAR(CREATE_ROW_TS) >=
        YEAR(CURRENT_TIMESTAMP - 3 YEARS)

CREATE VIEW ARCHTRANS AS
  SELECT * FROM TRANS_T
  WHERE YEAR(CREATE_ROW_TS) <
        YEAR(CURRENT_TIMESTAMP - 3 YEARS)

```

The reason for separating the data physically in the past was to ensure optimal performance for programs that were reading the physical files.

A business might have a business requirement to deprecate or drop an existing column in favor of a new column value. One typical example is that an application has stored a record added or a record last changed date in a character or decimal format, such as YYYYMMDD, but now the application wants to support more date formats like the European format of mm.dd.yyyy. Because the modernized file contains CREATE_ROW_TS, that value can easily be transformed to a date field. The view can also be used to ensure that the old column is not used by any application programs because it returns a default value instead of the actual value that is stored in the old create date column.

```
CREATE VIEW LOGVIEW (... , CRTDATE, NEWCRTDATE) AS
  SELECT ..., '00000000', DATE(CREATE_ROW_TS) FROM LOG_T
```

When the view is used by all application programs, the CRTDATE column can be dropped from the table.



Mobile, UI, and database modernization tools

There are many different techniques and tools that are available to modernize the UI or the database access of your applications without changing your business logic. This chapter describes some of the possible technologies and methodologies that are used by these techniques and tools. For some technologies, such as IBM Rational Open Access: RPG Edition (OA), you can easily modernize the UI interaction (including mobile) or the database access, but as you continue with your modernization efforts regarding the business logic, many more options become available to you.

10.1 Why you should consider a modernization tool

Tools are items that you use everyday to help you accomplish tasks. You use hammers to pound in nails, for example. This works well because the nail is firmly secured into the wood and holds together the intended components. But think about the process:

- ▶ Purchase the tool (the hammer is fairly cheap).
- ▶ Learn to hold the nail and swing the hammer without hitting your fingers, or missing the nail and hitting the wood and causing damage. (There are quality issues and damage to other areas because of lack of control.)
- ▶ You might find a tight spot where you cannot swing, but must make short taps. It might take you much longer to put in the nail, assuming you do. (You might not be able to accomplish the task with the existing tool.)
- ▶ You might have 1000 nails to secure. Using a hammer works, but it takes you a long time (productivity concerns).
- ▶ The components that are being secured can easily shift while you are securing the nail and then are no longer perfectly aligned (quality issue).

Is there a better tool to accomplish this task? If so, then you purchase a tool that improves speed and quality, and reduces your learning needs. Consider what happens when you use a different tool: a pneumatic nail gun.

- ▶ It costs more to procure the tool.
- ▶ A nail gun shoots in a nail correctly every time (quality).
 - The learning curve is smooth. Set the gun where you want the nail and pull the trigger.
 - You do not miss the nail as you would with a hammer. There is no extra damage to wood surface or to the operator.
 - Because of the speed of the process, the components tend to stay aligned.
 - The tool can be used in tight spots.
- ▶ Speed. You can secure hundreds of nails in a short time frame (increase productivity).

Both ways are valid, and both methods can be used. However, just like software development, do not ignore the potential benefits that modernization tools can bring to your environment.

10.1.1 Mobile and UI modernization tools

There are many options for mobile and UI modernization tools. ISVs, including IBM, have been creating solutions to help users quickly and easily modernize the user interface for their “green screen” applications for many years. As you might expect, some of these tools work better than other ones. Some offer different features, methodologies, and technologies.

First, consider the basic premise that most of the tools in the marketplace use, which is shown in Figure 10-1 on page 465.

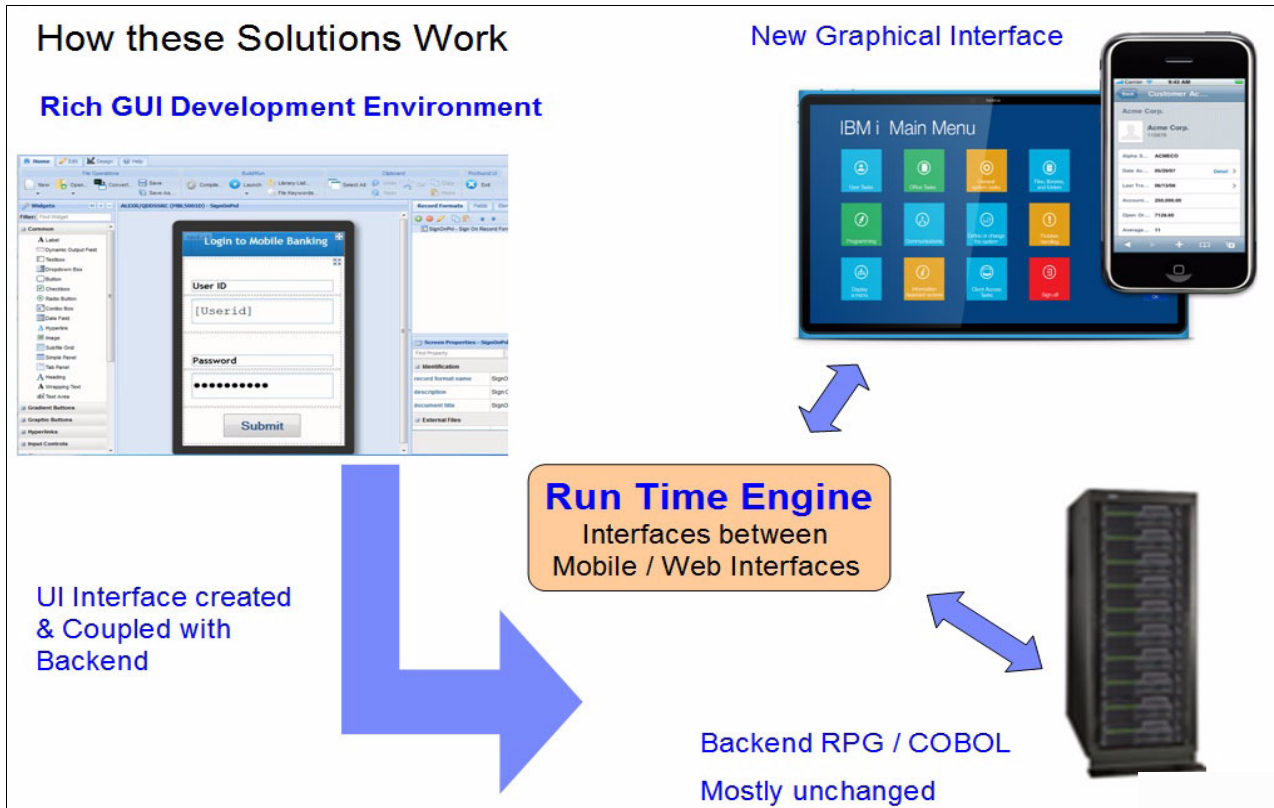


Figure 10-1 Modernization tools basic methodology

There are many different ISVs that provide solutions. All have their special feature that sets their solution apart from the other ones. Your task is to figure out what solution provides the most benefit for you. All of these tools have three primary components, as shown in Figure 10-1.

Rich GUI development environment

UI modernization tools have a GUI environment that the user or programmer can use to point at a back-end program or green screen and transform that green screen into a modern window. This is the development phase, where you can combine windows and fields into different layouts, using different, more modern window interaction widgets. For example, a green screen menu might be transformed into a drop-down box or a selectable list in a rich GUI interface. F keys (F1, F2, and so on) that are associated with a green screen table might be transformed into a drop-down list that is tied to every element in the table. These are the tools that can work with the technology that is being used (client, web browser, or mobile).

Runtime engine

After you create the windows for the relevant technology (client, web, or mobile), you must have an engine that dynamically builds the windows, gathers all the data, and processes the interactions. The *runtime engine* is often where there can be significant differences between solutions. Some solutions use a Java based runtime engine, some might be using a PHP-based runtime engine, and others might be using a more native solution that is running as part of an HTTP Apache server. Whatever the technology of the runtime engine, each of these engines is in the business of calling or accessing the back-end data and programs, gathering all the data, and transforming that content in to the user interface, displaying that interface, and managing any state information as needed.

New graphical interface

The result of the development interface and the runtime processing is the user interface. The user interface can range across a couple of different technologies. Some use a PC client technology, and others have a pure web interface. All technologies also are focusing on mobile in addition to these other options.

10.1.2 Why you should consider one of these solutions

The reasons your company might choose a solution can be different from the reason another company chooses a solution. Here is a list of some of the many reasons these types of solutions are considered:

- ▶ They use existing RPG/COBOL and DDS skills.
- ▶ Intuitive development environment.
- ▶ You can use your existing IBM i applications.
- ▶ You can transform in days, not years.
 - You make applications appear new.
 - You gain some time while you consider a longer-term solution.
- ▶ There is no need to learn new languages. Most companies want to focus on business, not being mobile developers.
- ▶ You can incorporate features of the mobile device into your back-end application.
 - Camera
 - GPS location
 - Maps
 - Others
- ▶ You write once, and then deploy to many devices.
- ▶ You leave the need to locate the devices to a third party.
- ▶ Cost effective.
 - Compare the cost of rewriting versus transforming.
 - Compare using tools to learning the technologies yourself.
- ▶ You can bring in new technology without having to start from scratch.

You might have tried one of many solutions in the past, and you might have encountered mixed results. Was the solution too slow? Did it make the green screen only display on a browser? Did the cost not justify the benefits of the tool when you considered its limitations?

If you encountered problems in the past, or heard about them, it is time to set aside your preconceptions and conduct a new evaluation. Most of the solution providers have continued to improve and adopt new technologies. The tools that were produced five or ten years ago are different from today's tools. Solutions today are using rich, web-based technologies such as JavaScript, Ajax, jQuery, or leading-edge mobile solutions. These newer solutions can help you transform an old green screen application so that the average user does not know that you did not rewrite the entire application. Each tool has countless tales of success. You can use these tools to move an entire green screen application (consisting of hundreds of screens) into a mobile-based solution in a matter of weeks.

10.2 IBM Rational Open Access: RPG Edition

For some, IBM Rational Open Access: RPG Edition (OA) is a new technology. For others, it is a new architecture or a new path. Defining OA is a difficult task because it can assist in refacing, refactoring, or new development.

OA was created to help avoid the limitations of the 5250 data stream. The RPG language has since its beginnings been tightly tied to the operating system and the 5250 data stream for all of its I/O activities. OA breaks this tight integration and opens RPG to a whole new world of I/O. OA enables you to interface with new devices and resources using native RPG code. The mechanism works by redirecting RPG I/O requests to a new program, which is called an OA Handler. The OA Handler can be written in any ILE language. Figure 10-2 show some examples of the many different ways that OA can be used.



Figure 10-2 Some examples of OA applicability

Consider the OA handler as a plug-in to the RPG run time. Its implementation is realized through an F-specification that indicates which OA Handler to use for the file. Any WORKSTN, PRINTER, or DISK files can be plugged in to an OA Handler. The **HANDLER** keyword is the only difference between using conventional RPG I/O devices and the devices that are controlled by OA Handlers. Example 10-1 shows the **HANDLER** keyword that is specified on a few of the different support specifications.

Example 10-1 Example of the HANDLER keyword for WORKSTN, PRINTER, and DISK files

Fmyfile1	CF	E	WORKSTN	
F				HANDLER('MYLIB/MYSRV-
F				(hd1Workstn)')
Fmyfile2	0	E	PRINTER	
F				HANDLER('MYLIB/MYSRV-
F				(hd1Printer)')
Fmyfile3	UF	A E	K DISK	
F				HANDLER('MYLIB/MYSRV-
F				(hd1Disk)')

With the **HANDLER** F-specification, all the file I/O is directed to and monitored by the OA Handler. Table 10-1 lists all the different operations that can be supported by an OA Handler.

Table 10-1 All I/O operations

CHAIN	FEOD	READE	SETLL
CLOSE	OPEN	READP	UNLOCK
DELETE	READ	READPE	UPDATE
EXFMT	READC	SETGT	WRITE

An OA Handler can monitor for all, or a subset of, the I/O operations that are available for the specified device type and take a specific action for each of them.

The point is that the OA Handler can provide all the infrastructure for new devices or resources and the RPG remains the same and uses conventional I/O operations. What does this statement mean? First, consider how the I/O interaction is handled in RPG. When an I/O interaction is called in the RPG code, without OA, the IBM i operating system is in complete control of the entire flow. Figure 10-3 shows an example of what this process might look like. You can see that all the data is stored in a program buffer. That buffer, which contains many things such as the data, state information, and pointers, is passed to the IBM i operating system. The operating system then transforms that content based on things such as the DSS into a user interface transaction. There are no entry points into this process. Before OA, if you wanted to intercept the data in this process, the only option was to capture the 5250 data stream that came out at the end of the process. There was no way to pass more content except for passing things in hidden fields, or specifying some additional side file that contained the extra things.

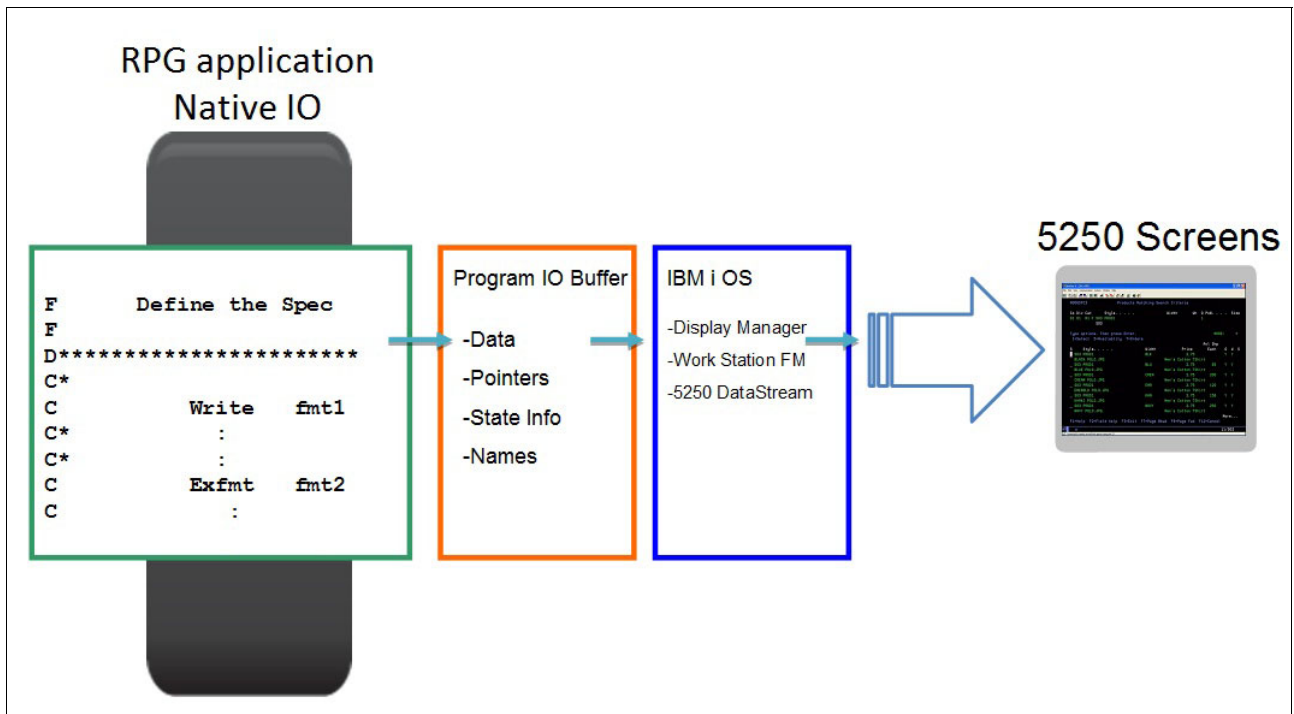


Figure 10-3 I/O flow from an RPG program to the user device before OA

With the addition of OA, the programmer can intercept the I/O transaction at a much earlier point in the flow. Basically, you get the opportunity to remove the IBM i operating system from the flow. This is a huge benefit. Figure 10-4 shows the new I/O flow when using the OA technology. Here, you can see that the OA Handler program is responsible for the work that the IBM i operating system did for the 5250 path. The handler, just like the OS, is given access to the program I/O buffer. This means you can control the UI interface from your RPG code.

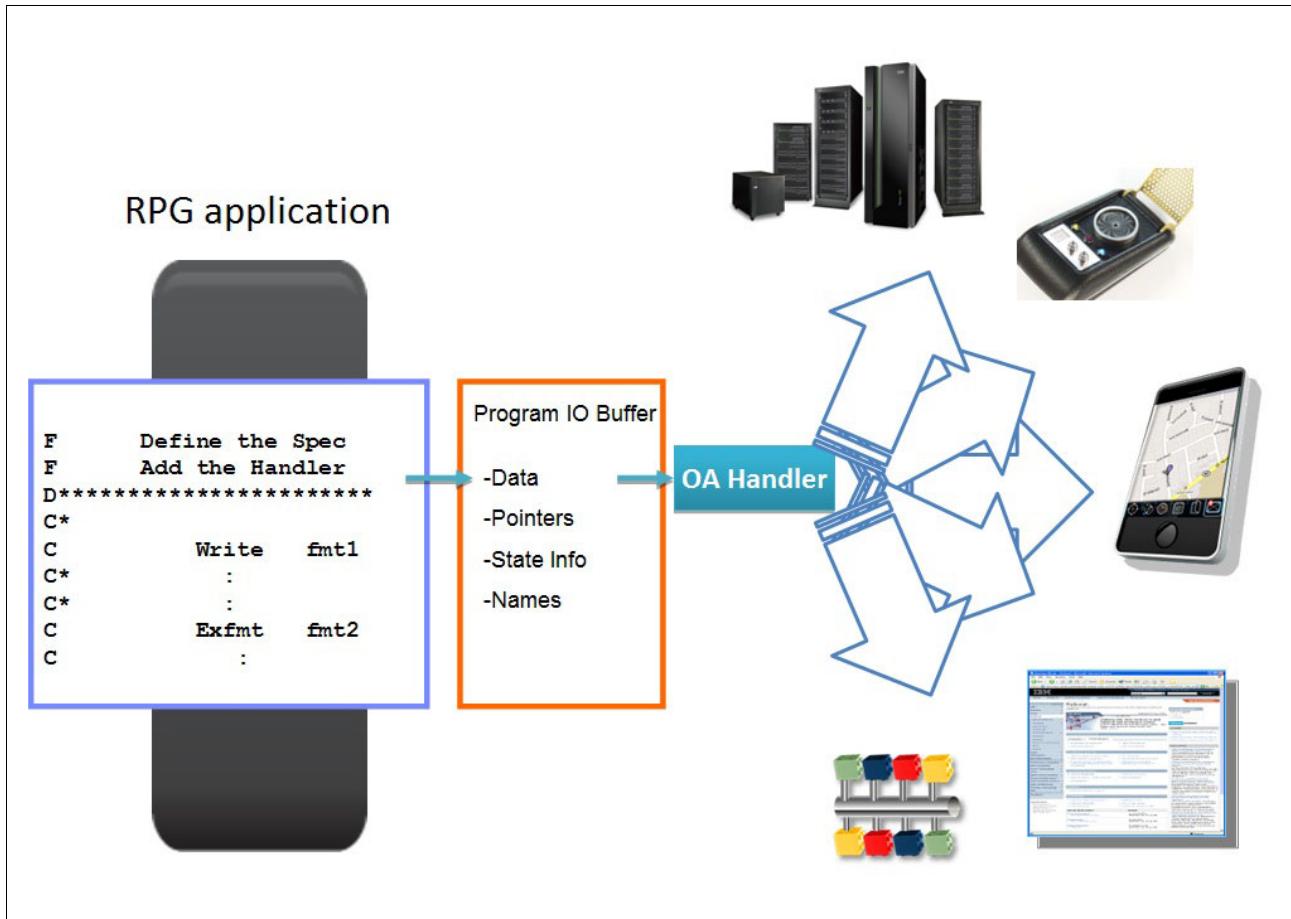


Figure 10-4 RPG I/O directed to the OA Handler

The major advantage of this technology is that you ultimately control new devices and resources directly in RPG by using the well-known RPG I/O model.

This control enables you to achieve the following goals:

- ▶ Capitalize on your RPG knowledge. For example, an RPG programmer can use their knowledge of disk file I/O to write programs that directly read and write IFS files. Alternatively, they can use their knowledge of display file I/O to write programs that interface to browsers or mobile devices.
- ▶ Create an effective bridge and collaboration with other programming languages and skills. For example, a team of .NET, JavaScript, or PHP programmers can work on the user interface side of the OA Handler and the RPG team can work on the RPG I/O and business logic side. The collaboration helps both teams to better understand the whole technology. You can rationalize the multi-programming language and multi-environment collaboration.

The RPG I/O model provides more than just control of the actual input and output operations to the device. It also provides for the marshalling and distribution of the data. This is the feature that most RPG programmers miss when forced to directly code to an API interface to control devices that are not normally supported by RPG.

When you access and control new devices and resources with OA, you can keep the business logic of your application in RP, so all the normal tasks, such as debugging, maintaining, and deploying your application, can be rationalized and simplified.

The user experience is richer and the programmer experience follows a “do no harm” methodology.

10.2.1 Architectural modernization

The way a program is bound to a file in a classic RPG program represents a monolithic system.

With OA, you can get the benefit of a multitier application without having to change your RPG business logic because OA decouples the I/O and can act between the RPG and the user interface (UI) or the database access. With the OA Handler, you can switch to another UI or change the behavior of the UI without changing the RPG business logic.

Multitier architecture

Here are the three tiers to consider:

- ▶ User interface (UI)
- ▶ Business logic
- ▶ Database

The advantage of a multitier architecture is to have the possibility to change one tier without affecting the others.

10.2.2 New control capabilities

OA enables the control of new elements and architecture within your application, as shown in Figure 10-5 and the following list:

- ▶ The I/O memory
- ▶ The UI metadata
- ▶ The communication channel
- ▶ The multitier capacity

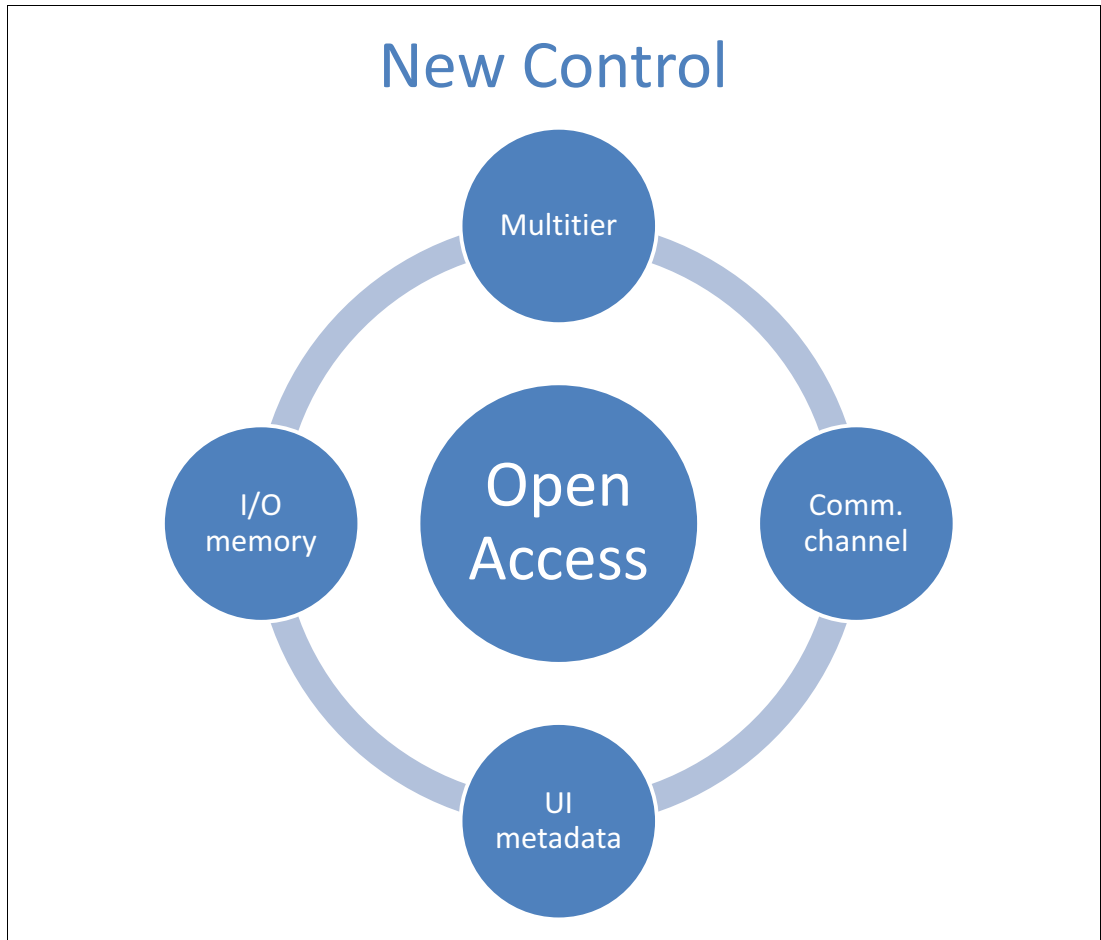


Figure 10-5 The new control capabilities of OA

Control of the full I/O memory

Every time a write record format is processed by the RPG program, the OA Handler receives the entire record content.

Complete control of the different record formats can be significant for display files.

With the 5250 workstation, limitations are imposed on the number of formats that can be displayed, the sequence of the write operations, overlays for record formats, and the size of the screen. Another limit is imposed on the total number of subfile rows and the number that can be shown on a screen. OA gives you the ability to see then entire data set and flow. Figure 10-6 on page 473 shows an analogy of before and after OA.

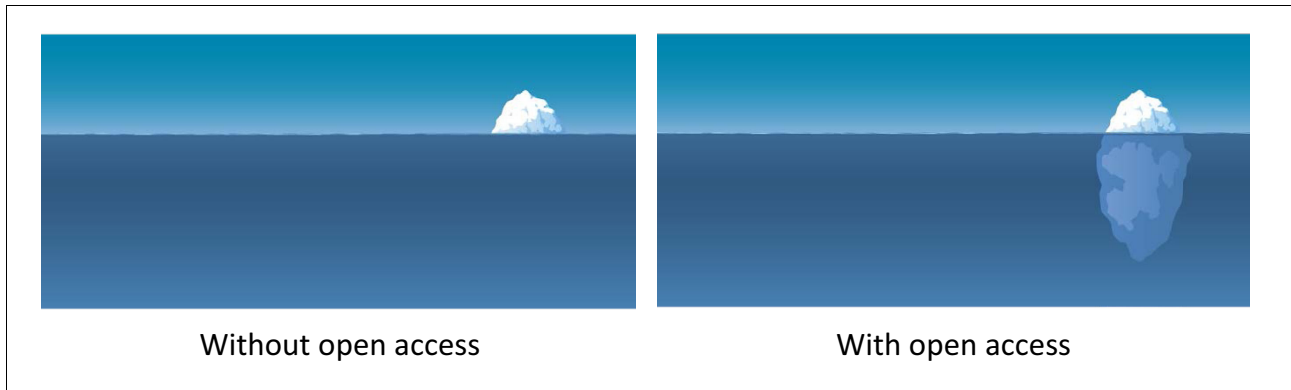


Figure 10-6 With 5250, you can see only the tip of the iceberg, but with OA you can control the whole iceberg

With OA, these limitations can be overcome. Any written format buffer is available in addition to any field value, including hidden fields. The OA Handler for a graphical UI can show more formats and more subfile rows. An OA Handler is also not restricted to reading only record formats that are written to the device.

Control of the communication channel

When an RPG program performs an I/O operation for a system file, a system data management function is called to handle the operation. When an RPG program performs an I/O operation for an OA file, the OA Handler is called.

The OA Handler can process the RPG I/O and direct it in to new channels and new protocols, as shown in Figure 10-7. The handler can access almost anything. Ultimately, it becomes your choice. The handler is important because it separates the UI from the application. It enforces a layering that enables you to easily move to a new UI methodology. Change your handler and move from a browser interface to a mobile interface.

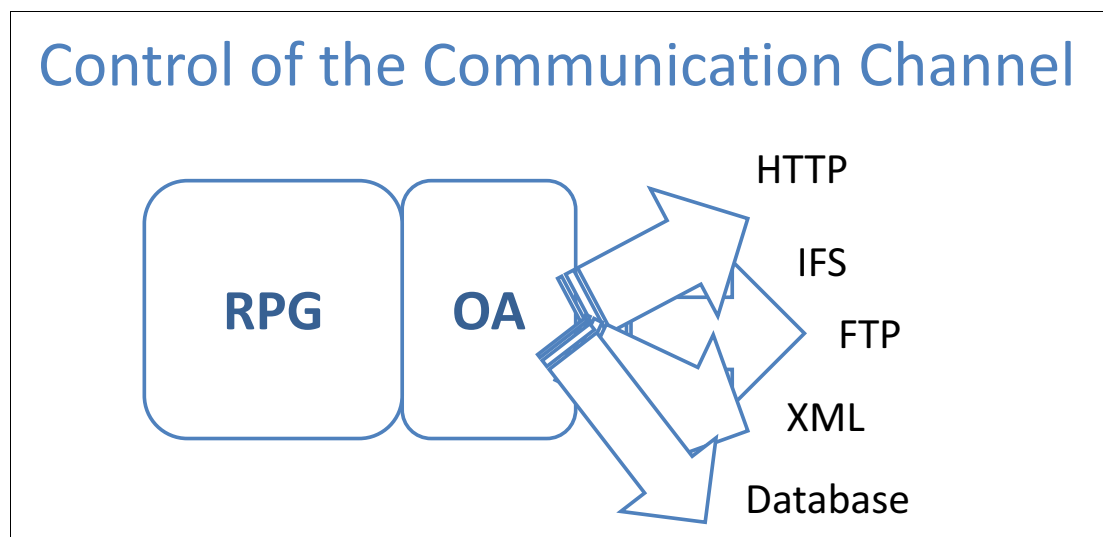


Figure 10-7 OA enables the control of new communication channels

Control of the multitier capacity

OA decouples the I/O end-to-end access, so it represents a multitier enabler, as shown in Figure 10-8.

With OA, you can change DB access or UI without changing your RPG business logic code.

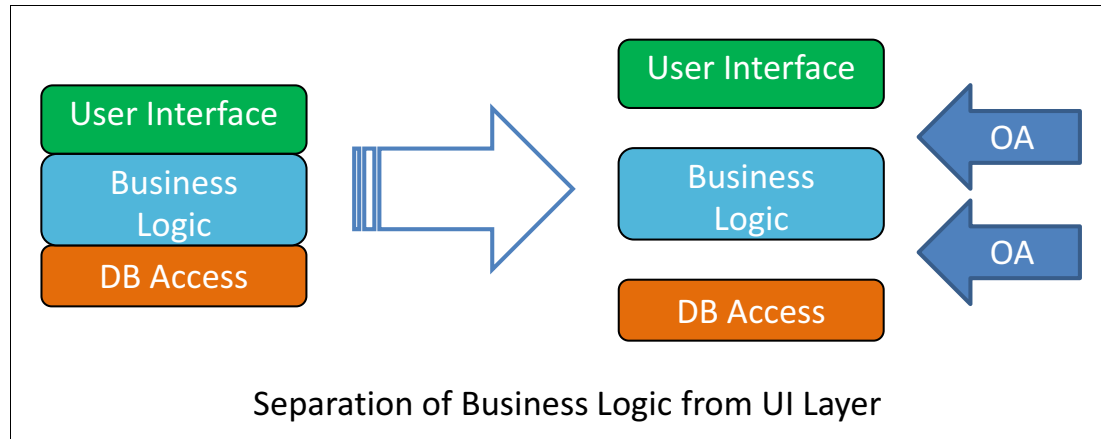


Figure 10-8 DB access and UI access are decoupled with OA

Control of the UI metadata

Metadata is extra data that is used to control your I/O fields on your new UI device. You can have metadata within the DDL or your DDS (or externally) that describes your file, or you can have metadata that describes each new element for the new device or the new resource to which your OA handler is connected.

Depending on the OA implementation you are using, you might need the metadata to be accessed from different processing languages or different tiers. With OA, you can entirely control the format, the extraction, and the location of the metadata. The OA developer has many choices for encoding metadata. XML is sometimes used to support multiple platforms and multiple use cases. JSON is popular for web browser solutions because of its small size. The remainder of this section describes metadata from an XML perspective. Understand that this can all be done in JSON (as demonstrated by some solutions).

XML is a cross-platform, industry standard to define metadata. It can be validated against a DTD. XML can also describe constants or literals in Unicode (DDS-based storage cannot). XML is a good candidate to store and share metadata for any OA implementation and extension. In addition, XML processing is directly supported by RPG's **XML-INT0** and **XML-SAX** operation codes.

Note: Metadata can be represented in any definition language. The most popular languages are XML and JSON.

Figure 10-9 on page 475 shows that DDS metadata and new metadata both can be stored in XML.

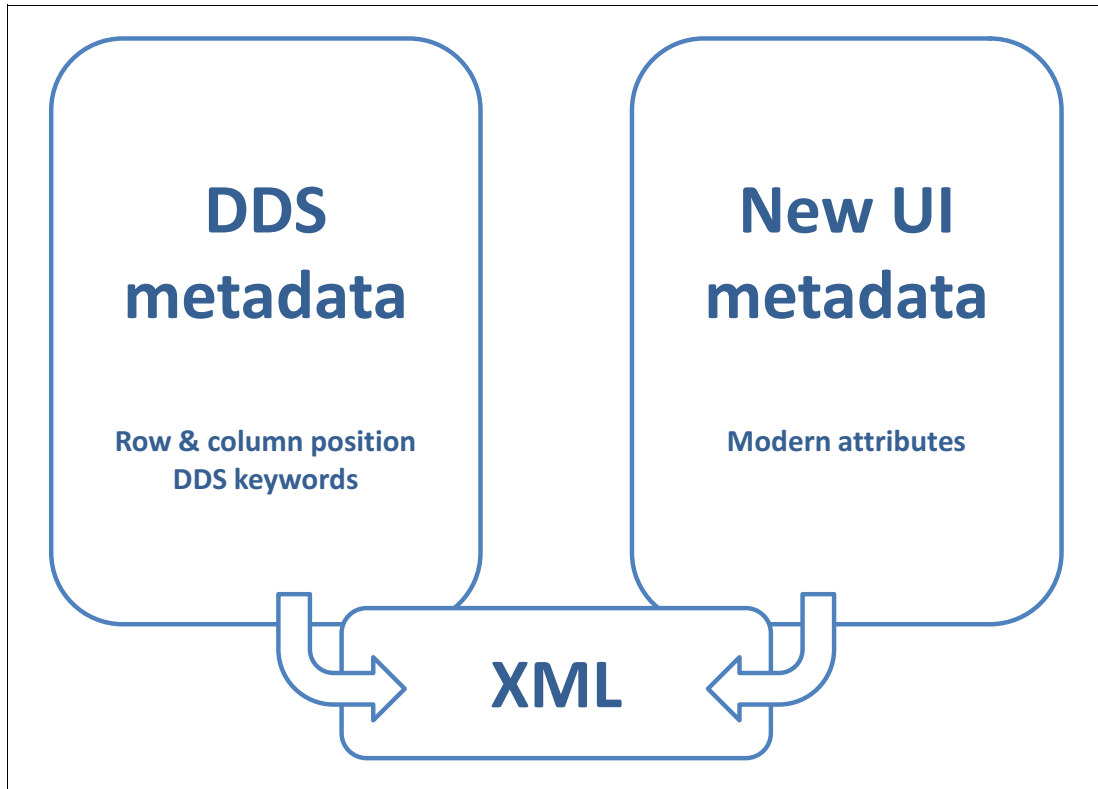


Figure 10-9 DDS metadata and new metadata can both be stored in XML

How to use XML for metadata from DDS and other sources

OA uses the name of the file for the OA Handler to reference and does not require the file to be present at run time. Therefore, the metadata does not need to be tied to the object file and you can generate preliminary metadata from different sources and store them anywhere.

10.2.3 User interface

The user interface is composed of two levels of information: the field buffer definition and the metadata definition. The metadata covers information such as the positioning (for DSPF and PRTF) and any attributes that are described through DDS keywords.

With OA, the metadata is not encapsulated in a system data management and the OA Handler is responsible for obtaining the metadata file. This process is what enables a multitier architecture to take place because the OA Handler can extrapolate the metadata dynamically or from an external repository that can be in any format.

Data field definition

The data field definition is the buffer definition that is wrapped by the RPG compiler for each record format. It can be composed of fields and indicators:

- ▶ Indicator name with length of 1 and Boolean type
- ▶ Field name with length and type

Note: OA can control the I/O buffer by field name or by format name.

Metadata definition

The metadata definition is any data about your I/O fields.

From a DDS perspective, it includes:

- ▶ Field positioning (row and column) for DSPF and PRTF
- ▶ All DDS keywords

From a UI perspective, it includes:

- ▶ UI attributes
 - Color
 - Font type
 - Bold
 - Underline
 - SuperScript
 - Position
 - Many more attributes
- ▶ UI properties
 - Error handling
 - Relationship to other fields

Open standard

Several years ago, a group made up of ISVs, industry experts, and IBM collaborated to build a common standard for defining metadata. This group and the standard that was created are called Open Access Metadata Open Standard (OAMOS), as shown in Figure 10-10.



Figure 10-10 OAMOS logo

Version 1 was announced at COMMON 2012 by IBM and other members of the OAMOS consortium. Additions and updated versions have been discussed and evaluated through the OAMOS approval college.

This standard includes DDS information and other metadata that describes the new UI device or resource.

The core objective of this open standard is to provide a simple common foundation that can be shared by OA implementations. This standard is designed to reduce lock in to a specific implementation and allows for the mixing and matching of multiple solutions to add value to one another. It also ensures that each implementation can support various device types in a simple manner.

Note: The OAMOS consists of an XML or JSON tagging convention for metadata description. To be OAMOS-compliant, your solution must use the format convention that is described in the OAMOS wiki or to have an import/export function to your format.

Using the standard makes it easier to work across multiple types of handlers from different suppliers. Developing or interacting with a handler is made much easier with a known standard that provides a transparent and extensible architecture. An important benefit of the standard is that it is simple to implement but does not limit solutions to specific technologies such as a printer or web browser. This implementation provides a foundation for dealing with today's requirements while also allowing flexibility to adapt to future innovations.

An example of the OAMOS code for a simple subfile control is shown in Example 10-2.

Example 10-2 OAMOS XML sample

```
<?xml version="1.0" encoding="UTF-8"?>
<fmt name="FMT1CTL" lib="MYLIB" dspf="WRKMOVIE" type="SFLCTL" mode="80"
recasc="EC1SFL" CCSID="37" X="1" Y="1" width="80" height="24" strRow="1"
endRow="8">
  <fmtKwds>
    <fmtKwd kwd="SFLCTL" value="EC1SFL" />
    <fmtKwd kwd="SFLSIZ" value="0100" />
    <fmtKwd kwd="SFLPAG" value="0013" />
    <fmtKwd kwd="CF03" value="03" />
    <fmtKwd kwd="OVERLAY" />
    <fmtKwd cond="92" kwd="SFLDSP" />
    <fmtKwd cond="91" kwd="SFLDSPCTL" />
    <fmtKwd cond="90" kwd="SFLCLR" />
    <fmtKwd cond="89" kwd="SFLEND" />
    <fmtKwd kwd="PRINT" />
    <fmtKwd kwd="HELP" />
  </fmtKwds>
  <inds>
    <ind name="*IN03" use="I" />
    <ind name="*IN89" use="0" />
    <ind name="*IN90" use="0" />
    <ind name="*IN91" use="0" />
    <ind name="*IN92" use="0" />
  </inds>
  <flds>
    <fld type="const" use="0" X="33" Y="1" width="16">
      <fldKwds>
        <fldKwd kwd="DFT" value="Work With Movies" />
        <fldKwd kwd="COLOR" value="WHT" />
      </fldKwds>
    </fld>
    <fld name="DATEFROM" type="L" use="B" X="25" Y="6" width="10">
      <fldKwds>
        <fldKwd kwd="DATFMT" value="*DMY" />
        <fldKwd kwd="MAPVAL" value="('01/01/40' *BLANK)" />
      </fldKwds>
    </fld>
    ...
  </fmt>
```

For more information about the OAMOS, see the following links:

- ▶ The Open Standard for RPG OA Metadata, found at:
http://ibmsystemsmag.com/ibmi/developer/rpg/oa_standard/
- ▶ OAMOS wiki, found at:
<http://www.IBMiOA.com>

10.2.4 Flow

With OA, the flow between the RPG and the OA Handler is as shown in Figure 10-11.

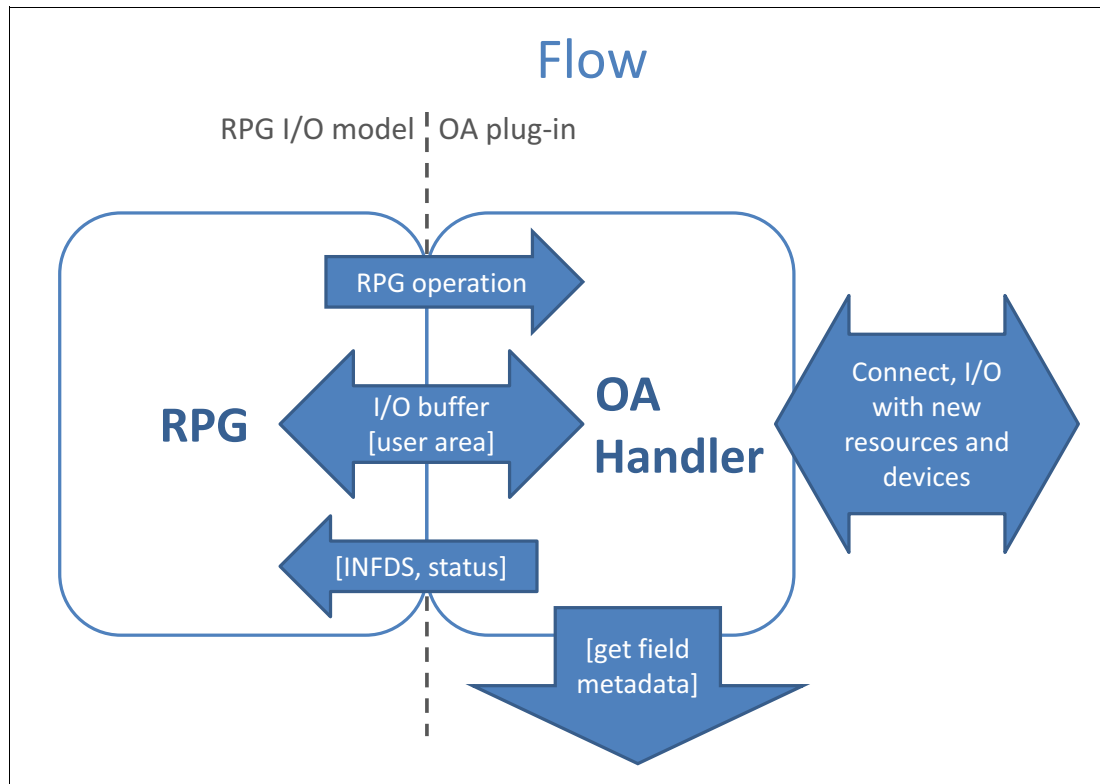


Figure 10-11 Global flow with OA

As you can see in Figure 10-11, everything always starts with an RPG operation that is passed to the OA Handler. The OA Handler then has control and can then get metadata from the file, connect, and exchange data with new resource or devices then return status or buffer data to the RPG. It is also possible to include a handler parameter to exchange additional information that is not available through an RPG file operation in a user area.

Between the RPG program and the OA Handler, the main articulations can be explained with three principal operations: OPEN, WRITE, and READ. In Table 10-2, you can find a description of these operations.

Table 10-2 Basic operations from the RPG to the handler

RPG to handler	Description
OPEN file	The file name is passed to the handler. The file object does not need to be opened or to exist at run time. It is used for reference. The handler sets up the mode to control the file buffer by field name values or by record format.

RPG to handler	Description
WRITE file-record	The handler receives the buffer and keys by field name value or by record format with data structures.
READ file-record	The handler passes the buffer by field name value or by record format with data structure.

Event-driven and stateless programming

Programmers who are familiar with UI programming tend to discount the 5250 programming model because it is not event-driven. However, with OA, you are free to design your programming architecture the way you want.

For example, an interactive program must start with a write or exfmt code (exfmt is equivalent to write then read). It is only possible to read record formats that previously have been written. With OA, your program can start with a read statement and this can allow you to emulate an event-driven programming environment.

RPG programs can be written as stateless programs, but heritage interactive RPG programs are stateful. To combine both designs, you can integrate a data queue or a message queue within your OA Handler.

10.2.5 Debugging

Debugging an OA program is the same as debugging any other program.

10.2.6 Converting existing programs to Open Access

To enable an RPG program for OA, all that is necessary is to add the **HANDLER** keyword on the F-specification that is related to the file and recompile the program.

For the handler, the level of complexity can vary upon the extension of the metadata you want or need to cover and the device or resource you want to control.

For existing programs, metadata might include DDS keywords to handle.

Physical or logical files use few DDS keywords. Therefore, creating a handler for it can be a fairly easy task.

For printer and display files, it is a complex task that might be simplified by limiting the DDS keywords that you want to integrate. This type of handler often is provided by an outside expert, such as a software tool vendor or business partner.

Extending converted programs to new features

With the control of the I/O memory, UI metadata, communication channel, and the multitier capacity, you can include new features in to your RPG without internal change (except the addition of the **HANDLER** keyword on the F spec). For example, for display files, you can add UI controls such as a “combo date” when a metadata signals that a field is of type L (date) or a combination list when the keyword **VALUE** is used or shows more subfile rows, allowing the OA Handler to acknowledge these interactions.

10.2.7 Creating programs with Open Access

This process presents some interesting possibilities because you see that when you are simplifying the DDS and extending the use of OA to rich UI components, you can create modern and rich applications by using RPG OA.

Here are some principles to consider first:

- ▶ DDS can be used without keywords.
- ▶ For the RPG, hidden fields are considered as input and output, so the RPG program can write and read hidden fields in addition to the UI.
- ▶ The RPG can write and read many formats. For display formats:
 - The workstation 5250 filters the formats that are displayed.
 - With OA, all written formats can be available (remember the iceberg).
 - With OA, you can read formats that have not been written. They can be initiated directly within the UI.
 - Many formats can be exchanged between the RPG and the UI in a single transaction.

Here are some examples of what you can achieve:

- ▶ Use hidden fields to pass data field but also metadata, such as a UI properties value (for example, the background color, image source, link URL, or combination list content).
- ▶ Lay out the formats that are sent by the RPG, vertically and horizontally, in any UI control, and use the GUI paradigm.
- ▶ Enable your RPG to control field data processing and also UI properties.

Using OA and RPG gives complete control to the RPG deliverable for controlling layout, color, and many other attributes for any UI solution, including browser or mobile.

10.2.8 Enterprise integration

OA is a natural extension of RPG and it is a perfect integrator between RPG and new device or resource access technology (Figure 10-12). OA can also rationalize the relationships between the different sets of skills that are required for a modernization project.

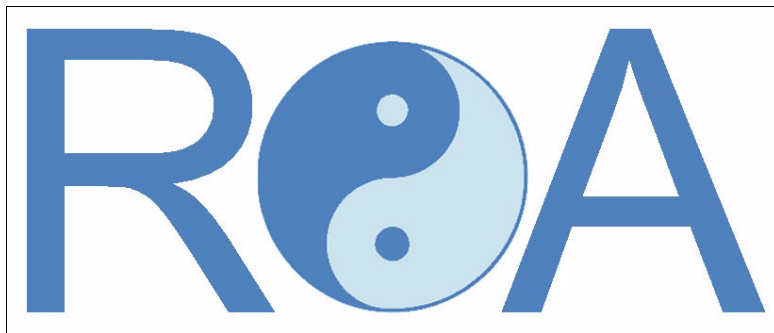


Figure 10-12 Integration with RPG OA and from RPG OA

For the runtime interactive programs, you might need to integrate the 5250 I/O flow with the OA I/O flow. For example, this process might be required if an OA program starts a non-OA program such as a system command (for example, `WRKSPLF` or `WRKACTJOB`) or any program that is not compiled with OA.

You also can have a non-OA program (for example, a 5250 menu or a sign-on screen or simply a program that is not compiled with OA) that starts an OA program.

In all of these cases, you have several levels of integration that you must consider. You have the capacity to start an OA program from a 5250 program and vice versa repetitively, and you have the capacity to run all in the same job and in the same UI session.

10.2.9 Open Access Handler example

This example uses a field-based OA Handler for a physical file and uses the open/close/write/read operations.

You can insert any code extension to use the file output and input values within the OA Handler, for example, to create or read an IFS document, a spreadsheet, or to start a web service.

The example code is made of three elements:

- ▶ Source member: MYHANDLER. The OA Handler (Example 10-3)
- ▶ Source member: MYRPG. The RPG using the Handler (Example 10-4 on page 483)
- ▶ Source member: MYFILE. The PF used by the RPG and the Handler, in DDS or DDL (Example 10-5 on page 484 and Example 10-6 on page 484)

Example 10-3 Source member MYHANDLER - The OA Handler

```
* Description: OA Handler example for PF/LF
*             with open/close/write/read operation
*
* Compilation:
*   crtrpgmod MYLIB/MYHANDLER srcfile(MYLIB/qrpglesrc)
*   crtsrvpgm MYLIB/MYHANDLER module(MYLIB/MYHANDLER)
*             export(*all)
*
*=====
H nomain
H option(*srcstmt)
*=====
D RcdFmti      E DS          extname(MYFILE:*input)
D              qualified
D RcdFmto      E DS          extname(MYFILE:*output)
D              qualified
*=====
/copy QOAR/qrpglesrc,qrnopenacc
*=====
D OADISK      pr          extproc('OADISK')
D  info      likeds(QrnOpenAccess_T)
D disk_OPEN...
D              pr          n
D  info      likeds(QrnOpenAccess_T)
D disk_CLOSE...
D              pr          n
D  info      likeds(QrnOpenAccess_T)
D disk_WRITE...
D              pr
D  info      likeds(QrnOpenAccess_T)
D disk_READ...
```

```

D          pr
D  info          likeds(QrnOpenAccess_T)
*=====
P OADISK      b          export
D          pi
D  info          likeds(QrnOpenAccess_T)
/free
  if info.rpgOperation = QrnOperation_OPEN;
  if not disk_OPEN(info);
    dsply ('File ' + %trim(info.externalFile.name)
          + ' of type ' + %trim(info.RpgDevice)
          + ' not handled.' );
    info.rpgStatus = 1299;
    return;
  endif;
elseif info.rpgOperation = QrnOperation_CLOSE;
  if disk_CLOSE(info);
  endif;
elseif info.rpgOperation = QrnOperation_WRITE;
  callp disk_WRITE(info);
elseif info.rpgOperation = QrnOperation_READ;
  callp disk_READ(info);
else;
  dsply ('Unhandled operation for disk file '
        + %char(info.rpgOperation));
  info.rpgStatus = 1299;
endif;
/end-free
P OADISK      e
*=====
P disk_OPEN...
P          b
D          pi          n
D  info          likeds(QrnOpenAccess_T)
/free
  // info.RpgDevice specifies the device
  // (D=disk, U=workstn, P=printer)
  if info.RpgDevice <> 'D';
    return *off;
  endif;
  // process the I/O by field name
  //info.useNamesValues = '1';
  return *on;
/end-free
P          e
*=====
P disk_CLOSE...
P          b
D          pi          n
D  info          likeds(QrnOpenAccess_T)
/free
  return *on;
/end-free
P          e
*=====

```



```

P disk_WRITE...
P          b
D          pi
D info          likeds(QrnOpenAccess_T)
D output       ds          based(p_output)
D          likeds(RcdFmto)
/free

    p_output = info.outputBuffer ;

    // Now the DS output is fulfilled and ready to be used.
    // You have output.MYFLD01
    //          output.MYFLD02
    //          output.MYFLD03
    // Add your code here:
    //

    return;
/end-free
P          e
*=====
P disk_READ...
P          b
D          pi
D info          likeds(QrnOpenAccess_T)
D input       ds          based(info.inputBuffer)
D          likeds(RcdFmti)
/free
    // Add your code here to complete all your INPUT field
    // Value in the DS input
    // For example:
    input.MYFLD01 = 22440;
    input.MYFLD02 = 'Buongiorno';
    input.MYFLD03 = 'Good morning mate';

    // If there are no values to return seton info.eof
    // info.eof = *on;
    //

    return;
/end-free
P          e

```

Example 10-4 Source member MYRPG - The RPG using the Handler

```

*
* Compilation:
*   crtbnrpg pgm(MYLIB/MYRPG)
*
H dftactgrp(*NO)
H option(*srcstmt)

FMYFILE    IF A E          DISK    extfile('MYFILE')
F          handler('MYLIB/MYHANDLER-

```

```

F                                (OADISK)')
/free

// set fields values and write the file record
FLD01 = 12345;
FLD02 = 'abc';
FLD03 = 'Heureux qui comme Ulysse conquit la toison d''or';
write myrec;

// Clear all field value
clear myrec;

// read the file record
read myrec;

*inlr = '1';
/end-free

```

Example 10-5 Source member MYFILE - The PF used by the RPG and the Handler - DDS version

```

*
* Compilation:
*   crtpf file(MYLIB/MYFILE)
*
A      R MYREC
A      FLD01      5  0
A      FLD02      30
A      FLD03     100

```

Example 10-6 Source member MYFILE - The PF used by the RPG and the handler - DDL version

```

CREATE TABLE MYLIB/MYFILE (
  FLD01 DECIMAL(5, 0) NOT NULL DEFAULT 0 ,
  FLD02 CHAR(30) NOT NULL DEFAULT '' ,
  FLD03 CHAR(100) NOT NULL DEFAULT '' )

RCDFMT MYREC;

```

In the article found at the following website, you find a description about how to create a format-based handler that allows multiple programs, which are updating the same table with a common format and unique key, to take advantage of a single generic SQL **UPDATE** statement. This statement uses extended indicator variable support.

<http://www.ibm.com/developerworks/ibmi/library/i-roaforsql>

10.2.10 Conclusion

RPG is universally recognized as the preferred language to process business logic on the IBM i.

OA allows the business logic in RPG to drive any UI and makes RPG a multitier development environment. An open standard is available for the description of advanced UI information in a published standard methodology. RPG has all the means for developing modern applications and supporting modern devices (Figure 10-13 on page 485).

Its features include:

- ▶ Multitier
- ▶ Centralization of the logic and native control of new resource or device
- ▶ Open standard

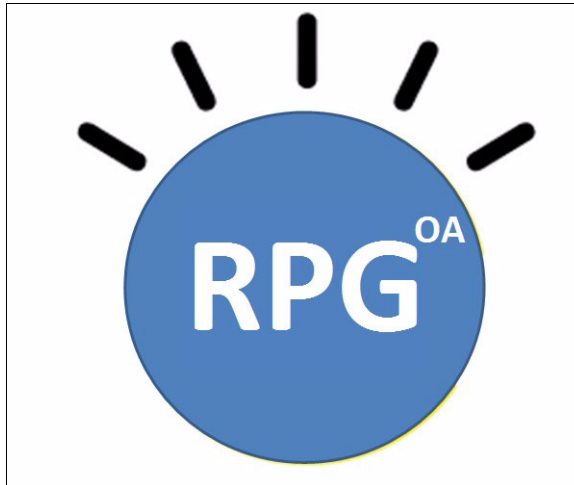


Figure 10-13 *RPG is a smart modern business language*

For more information about OA, PTFs, and coding documentation, you can access the developerWorks wiki at the following website:

<http://ibm.co/1f4wFwY>

10.3 5250 data stream interpretation

Before the invention of the OA methodology, the only option was to read the 5250 data stream and transform that data into a new rich and modern interface. This technology is still used by many solutions today with great effectiveness. There are still valid reasons to use 5250 data stream today, for example, the COBOL language does not have an OA method. Also, for applications where you do not have the source, you can still put a modern web or mobile front end on these applications by simply catching the 5250 data stream.

10.3.1 How the 5250 data stream method works

Much of what is described in the OA methodology applies to this support. The significant difference is that the solution runtime engine has access only to the 5250 data stream. Any additional content, or metadata, is available only through external means.

In Figure 10-14, you can see that the runtime engine is a Java engine that captures the output from the 5250 Telnet server. In this example, the runtime engine is enabling multiple output options, including web, mobile, and even web services.

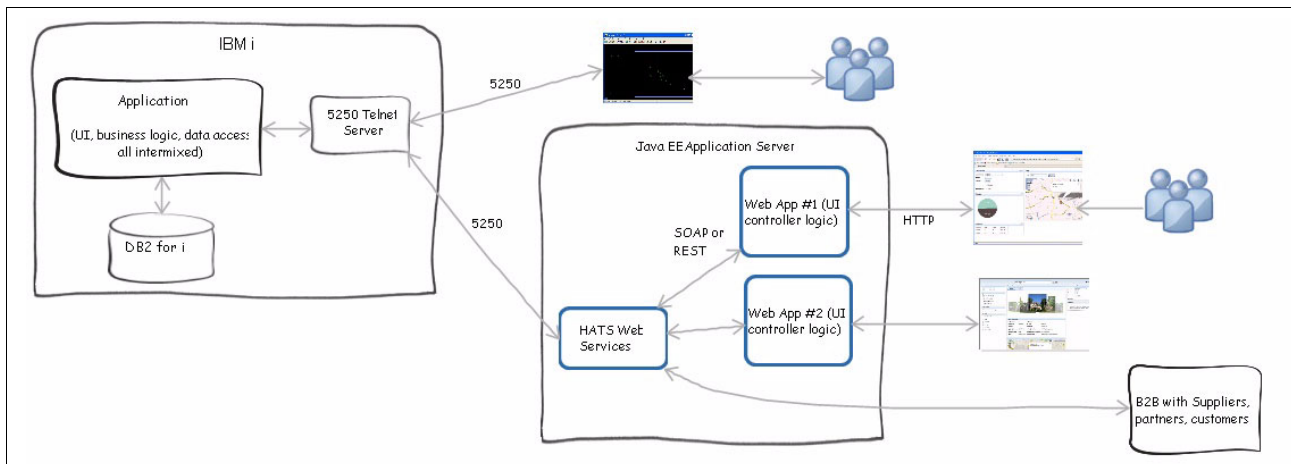


Figure 10-14 5250 data stream capture flow

This method is often referred to as *refacing*. Here are some of the reasons to consider this type of solution:

- ▶ Provide web and mobile access to existing applications quickly
- ▶ Low risk tolerance
- ▶ Acceptable for a host application to constrain how applications flow
- ▶ Keep 5250 access intact for power/back-office users
- ▶ Applications cannot be modified (lack of source code, lack of skills, too risky, and so on)
- ▶ Reduce IT operational costs that are associated with maintaining multiple emulators/versions across desktops
- ▶ Advanced web development skills are in short supply

10.3.2 IBM Rational Host Access Transformation Services

IBM has been in the 5250 data stream transforming business for a long time. With IBM Rational Host Access Transformation Services (HATS), you can create web applications, including portlets, and rich-client applications that provide a GUI for your 5250 applications that are running on IBM i. HATS applications can access 5250 applications without requiring Online Transaction Processing (OLTP) capacity. You can also create service-oriented architecture (SOA) assets using web services that provide standard programming interfaces to business logic and transactions that are contained within character-based 5250 applications. Data from virtual terminal (VT) emulation screens can also be accessed.

HATS applications can be given a modern appearance. HATS web applications, including portlets, can be developed with an interface that matches your company's web or portal pages, and your users can access them through their web browsers or they can continue to access the application using the 5250 interface. HATS web applications can also be developed to provide access from mobile devices, such as cellular phones, data collection terminals, and personal digital assistants (PDAs).

HATS rich-client applications can be developed to run in an Eclipse Rich Client Platform (RCP) implementation, in IBM Lotus® Notes®, or in the IBM Lotus Expeditor Client to provide native client applications that are targeted for a user's desktop.

HATS has two components:

- ▶ The HATS Toolkit is a set of plug-ins for the Eclipse-based IBM Rational Software Delivery Platform (Rational SDP).

The HATS Toolkit enables you to develop new applications, including portlets, a step at a time, previewing and saving each change that you make. Over time, or as quickly as you like, you can streamline your HATS application, making it easier to use than the host applications whose data it presents, and possibly moving functions from the host applications into your HATS applications. The development process for building HATS web and rich-client applications is similar.

After you have developed a HATS application, you deploy it to a production runtime environment.

- ▶ The HATS runtime code runs as part of a HATS application that is deployed in a production runtime environment, IBM WebSphere Application Server, IBM WebSphere Portal, or rich client platform. The users interact with the HATS application through the HATS GUI and data is sent back and forth between the user and the host application.

Figure 10-15 shows the HATS flow and options.

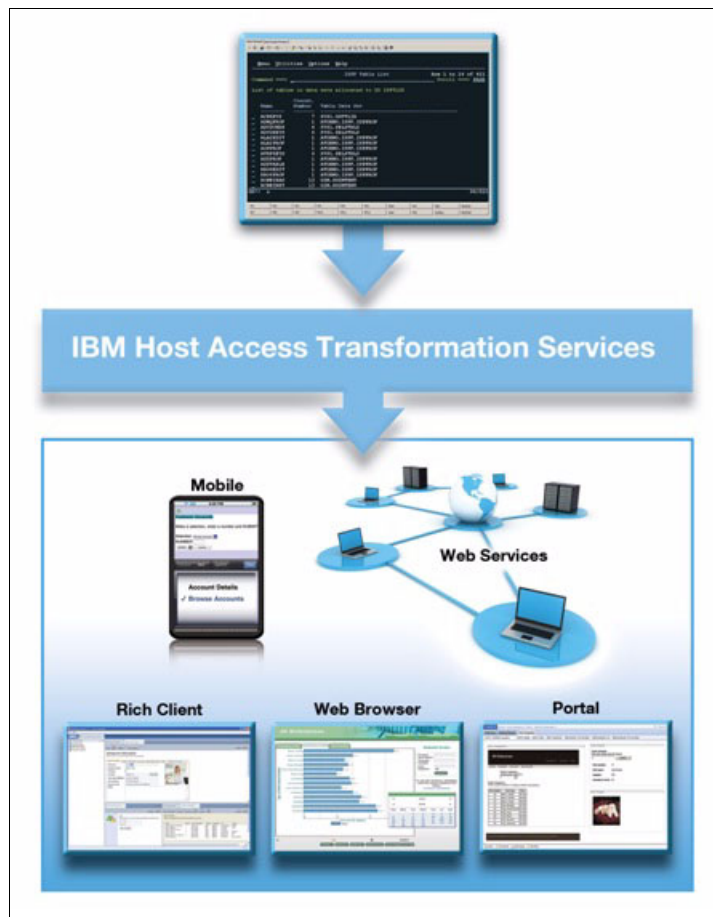


Figure 10-15 HATS flow and options

HATS provides a rich set of options and benefits for many devices:

- ▶ Rich Client
 - Integration at the desktop with other Eclipse-based applications
 - Client-side processing
 - Rich set of user interface widgets
 - Supports Lotus Expeditor deployment
 - Simplified specification for rich-client type-ahead support
- ▶ Browser
 - Zero footprint
 - Pure HTML
 - Access through your favorite browser, including Internet Explorer and Firefox
 - Extend to new users for drive new revenue or achieve cost reduction with internet self-help
- ▶ Mobile
 - Access host applications from mobile devices
 - Windows Mobile, iPhone, iPod Touch, and iPad
- ▶ Portal
 - Integration at the glass
 - Inter-portlet communication
 - JSR 168 and 286 compliant
- ▶ Web services
 - Build reusable transactions
 - Expose host business processes as standard web services
 - Provide controlled access to vital host applications and host data

For more information about HATS, see the following website:

<http://www.ibm.com/software/awdtools/hats/index.html>



EGL

For application modernization, there are many tools and approaches to enable developers to develop applications and modernize their existing applications and data. This publication highlights that fact as it covers many tools, depending on which approach or approaches you take to accomplish application modernization.

How does a staff learn all the various technologies? Especially if the staff is small?

For example, for traditional but experienced developers, many of the approaches require a long and sometimes impossible learning process for skills in object-oriented design and languages such as Java or JavaScript. For all developers, the need to know and use an ever-growing and evolving set of technologies for web services, Java Platform, Enterprise Edition, web, mobile, and others, are becoming more demanding. The inherent complexity in these technologies contributes to a lack of productivity in delivering the required systems to the business.

EGL uniquely solves the problem through a single technology, language, and toolset that allows the following actions:

- ▶ Efficiently allows you to build modern and complete “end-to-end” applications (user interface through business and database logic)
- ▶ Uses your existing IBM i assets, such as databases, programs, and people
- ▶ Can be quickly learned by both traditional and new programmers
- ▶ Can increase productivity for business application development

EGL is a lightweight programming technology that is designed and created by IBM to meet the challenges of modern development by providing a common language and programming model to build full-function business applications regardless of where the code is deployed. This includes business and database logic running native IBM i and Java Platform, Enterprise Edition application servers, such as WebSphere Application Server or Tomcat, and user interface implementations running in browsers. EGL uses proven, existing technologies, frameworks, and platforms, and easily enables integration with non-EGL technologies, such as RPG.

EGL is used by IBM i customers and IBM Business Partners to create web applications that access existing IBM i databases, RPG programs, or both. It is easily learned by traditional RPG developers because of its familiar procedural programming style, easily allowing access to web page and services development. For programmers more familiar with the newer technologies, it provides higher productivity compared to other technologies through a higher-level programming paradigm.

11.1 A closer look at EGL

At the most basic level, EGL is a modern programming language that is designed to help business-oriented programmers use the benefits of IBM i, Java, Java Platform, Enterprise Edition, mobile, and browser platforms without having to learn all the low-level details. EGL hides the technical details of the deployment platform and associated middleware programming interfaces, which enables the developer to focus on the business problem instead of the underlying implementation technologies. The language encompasses many features and common language elements:

- ▶ The ability to create Web 2.0 or rich internet applications (Rich UI).
- ▶ The ability to create mobile applications (running within a browser).
- ▶ The ability to create web services (both REST- and SOAP-based).
- ▶ The ability to work with a large list of data suppliers, including DB2, Oracle, SQL Server, and files. This includes tables, files, and data queues on IBM i.
- ▶ Usage of Rich Data Types, both primitive and complex.
- ▶ High-level language elements such as:
 - Automatic casting between mixed data types during operations
 - High-level abstractions and language verbs, such as “get” to perform database and file I/O
 - Full exception handling
 - Single specification of data validation, which can be reused
- ▶ A long list of built-in libraries, including ones to help with date, string, math, and XML manipulation.
- ▶ The ability to call out and integrate with other technologies such as Java, RPG, COBOL, and stored procedures.
- ▶ Standard text-based user interfaces for 5250 applications and printing.

From this common, high-level language, EGL can translate the user interface or business logic into a language that is appropriate for the deployment platform, including JavaScript, Java, or COBOL, as shown in Figure 11-1 on page 491.

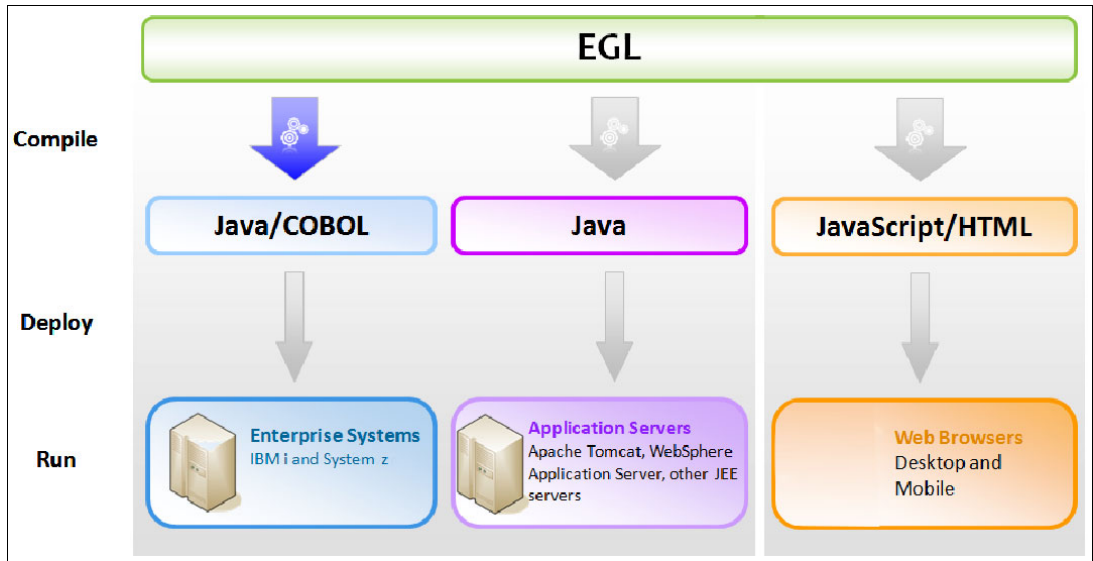


Figure 11-1 EGL language by platform

The broad language capability also provides great versatility in the types of applications and application architectures that can be created and the platforms in which the applications can be deployed, as shown in Figure 11-2.

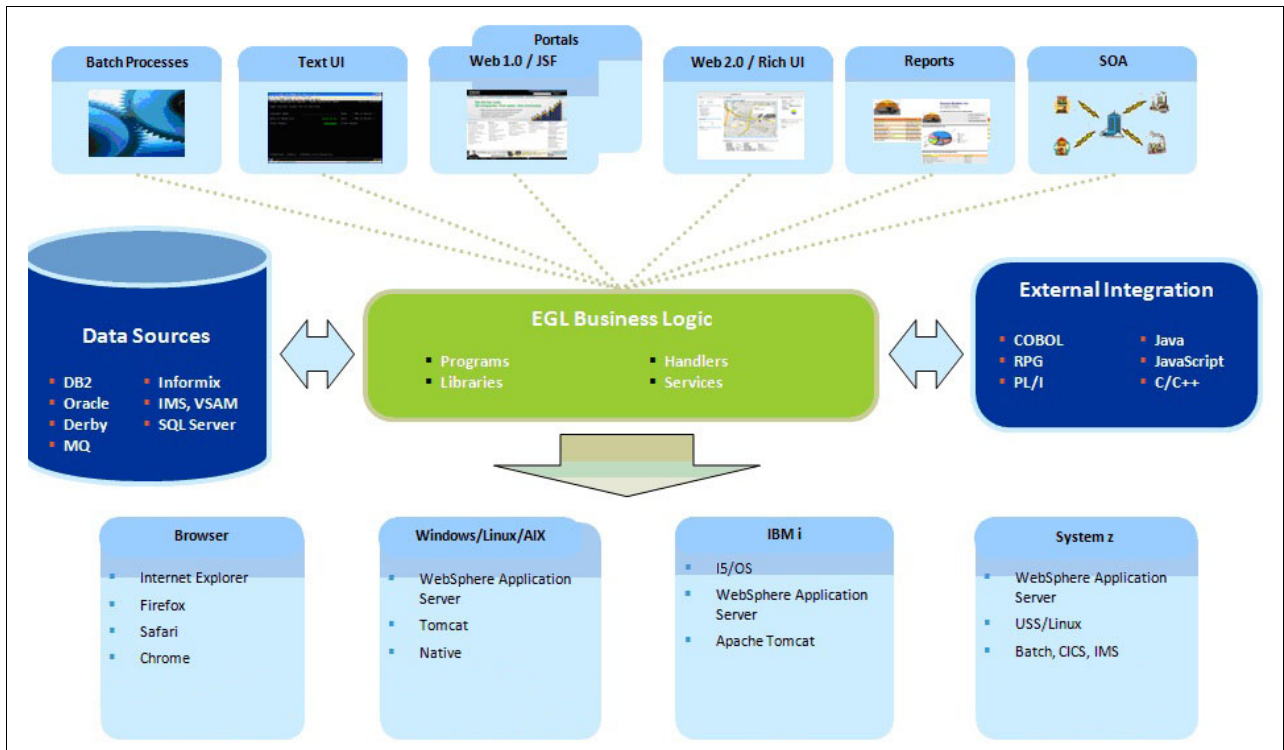


Figure 11-2 EGL versatility

At a broader and more comprehensive definition, EGL is a language and a highly productive development environment. The language and surrounding tools are based on the Eclipse-based programming workbench within the IBM Rational Software Development Platform and provide a common, integrated workbench for web, Web 2.0, SOA, data-oriented, and 5250 UI development. Here are the productivity features that are provided:

- ▶ Smart editors with content assist, code completion, refactoring, and predefined code templates
- ▶ Visual Editors for web pages (Rich UI), mobile pages, and Text UI (5250) development and design
- ▶ Integrated and end-to-end source level debugging, including web page layout and logic and the business and data logic
- ▶ SQL visualization and editing
- ▶ Services creation through wizards for database tables (see the following sections) and RPG programs
- ▶ References and declaration tools
- ▶ Integration with source control and change management systems, such as Rational Team Concert, to manage EGL development projects

Specifically, the EGL technology is included within the Rational Business Developer product. All versions can be used to develop for the IBM i. The latest version is Rational Business Developer V9. Rational Business Developer is also bundled within other IBM Rational IDE products, including Rational Developer for i V9.0, RPG and COBOL + Modernization Tools, EGL edition, and Rational Developer for the Enterprise V9.0. All the products allow EGL developers to target IBM i for deployment or for integration with IBM i resources.

11.2 EGL modernization options for IBM i

Consider what EGL can do specifically for IBM i.

As already stated, EGL has many capabilities to modernize IBM i based applications and technology that can be used at any point in the application architecture. For example, as Figure 11-3 on page 493 shows, EGL can be used for the following things:

- ▶ Creating web services for business and database logic with direct access to an IBM i database
- ▶ Creating web services with logic to start existing RPG programs or other languages through a remote call
- ▶ Creating an interface for RPG programs that allow that RPG program to be started directly as a web service (called an External Type HostProgram in EGL)
- ▶ Creating Web 2.0-based rich user interface (RUI) web pages, which run as JavaScript within the browser
- ▶ Creating business and data logic deployable as ILE COBOL on IBM i
- ▶ Creating batch/5250 programs when a need still exists

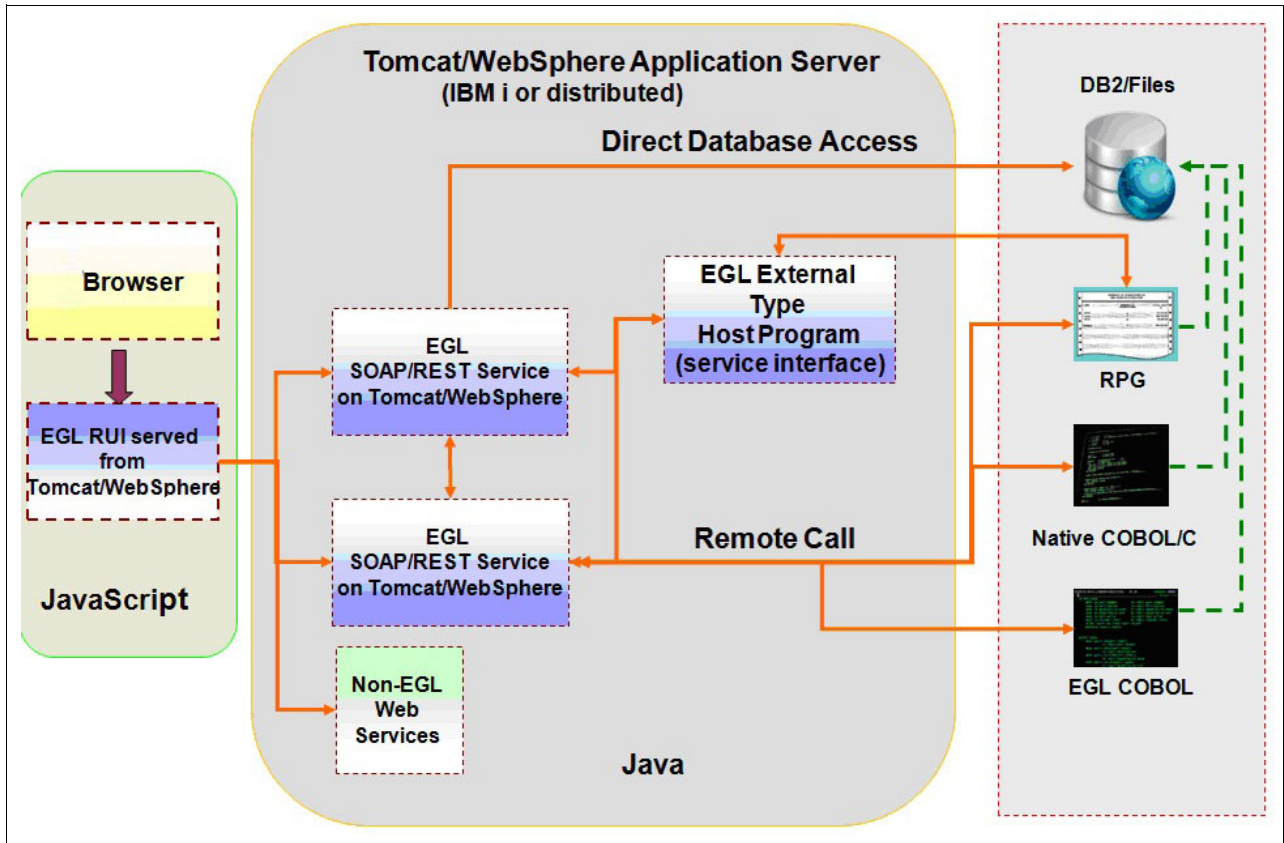


Figure 11-3 EGL application modernization options

11.2.1 Modernization through access to existing data

If the modernization approach is to build applications that access existing databases, as shown in Figure 11-4 on page 494, EGL provides a “Data Access Application” wizard that creates a complete web service with pre-built Create, Read, Update, and Delete functions for a particular table in a library. This service can then be started by an EGL rich user interface (RUI) to present the data to the user. More details about the EGL RUI are covered in 11.5, “EGL: From the database to the web” on page 507. It shows the steps that are needed to create and build both a data access service and build an EGL RUI to interact with it.

At run time, the EGL RUI web page runs as JavaScript within the browser. The service normally runs a Java application within an application server and uses the JDBC driver in the Java toolkit from the IBM i (JT400 Toolkit) to access the tables in an IBM i library. See the execution portion of Figure 11-4 on page 494. The JavaScript and Java are generated by EGL automatically. As a programmer, you need to know only the EGL language.

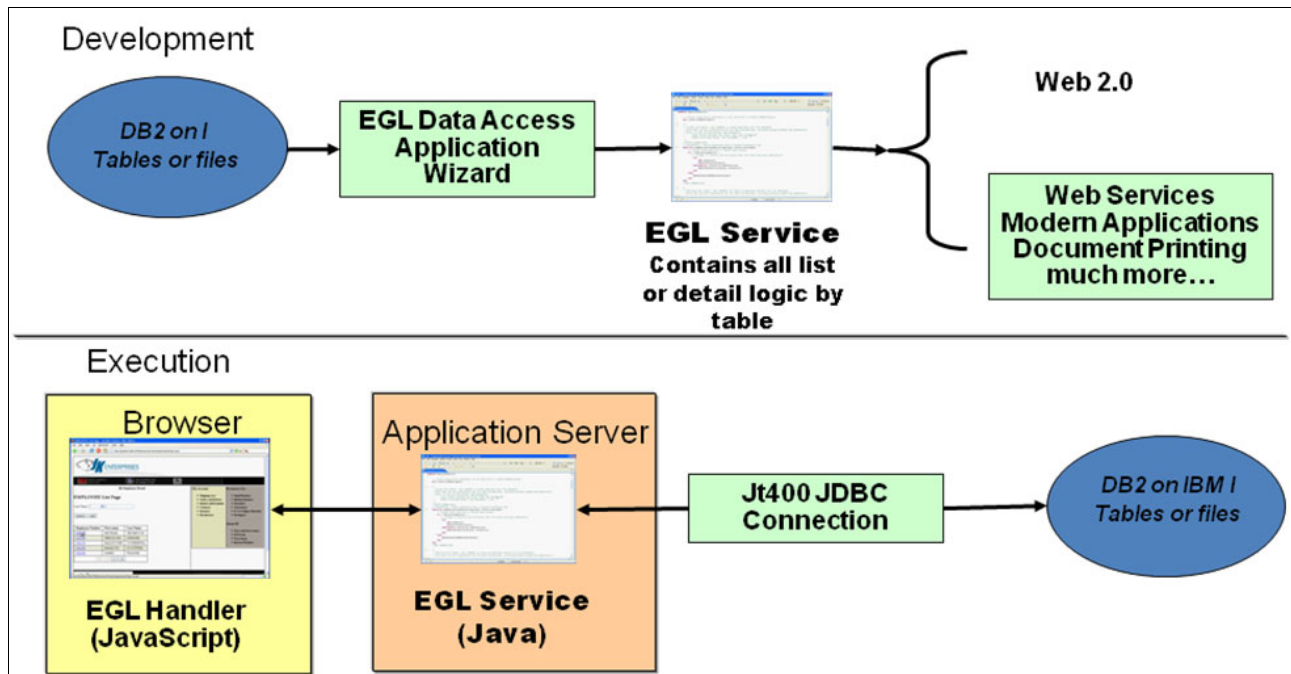


Figure 11-4 Modernization through a new application that accesses existing data

11.2.2 Modernization through reuse of existing RPG programs

If the application modernization approach is to reuse existing RPG programs that are running on the IBM i, then EGL can be used to create a service interface, which in turn can access the RPG program. For example, you can create EGL web services to do validation and then reuse existing RPG programs for the business and data logic. You can also create an EGL service to coordinate multiple calls to the existing programs on the IBM i.

One way that EGL enables access to an RPG program is through a wizard, which creates an EGL interface and records from an existing RPG callable program. This is named an EGL type “HostProgram”. In the interface, the information, such as what system and library contain the RPG program, is specified, and this is used at run time to start the program.

The other way is to use standard EGL “call” statement syntax within an EGL service. EGL provides a specification file that is called a linkage part, which provides the necessary information about things such as what system the RPG program runs on, what library it is in, and how to convert the data to EBCDIC. This allows the programmer to not be concerned with the deployment information and have the freedom to change it at run time or generation time, versus having to change actual code.

Both ways use the JT400 Toolkit's remote program call functions to make the call between the service that is running on an application server and the RPG program that is running on a native IBM i. For more information about the HostProgram, see Figure 11-5 on page 495; for the steps to code and set up a remote call to the IBM i, see 11.6, “EGL: Calling external programs” on page 536.

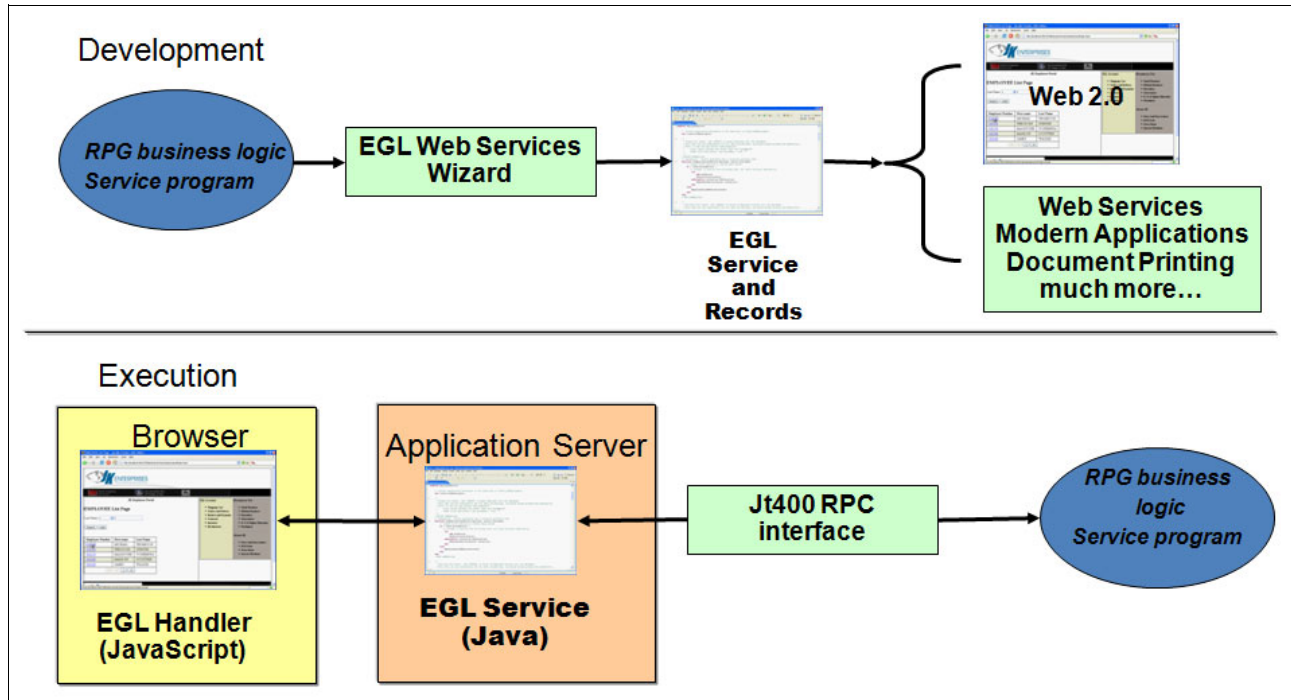


Figure 11-5 Modernization by reusing existing RPG

11.3 A brief look at the EGL language

One of the productivity features of EGL is its common language and syntax for building all parts of an application, including fulfilling modern requirements such as Web 2.0-based web pages, web services, and data access. Also, as mentioned before, the language is easily learned by traditional and new programmers.

This section shows the basic syntax of key EGL building blocks in order for you to get a feel for how easy the language is to use and learn.

The basic structure of all EGL logic is the same. The term logic means something that you write that is started in the system to run your business rules or perform data access. For EGL, this includes programs, services, and libraries.

EGL programs are traditional executable programs that are started with or without parameters and typically have one entry point. They can be a main program or a callable subprogram and either type can be used in batch or online applications.

EGL services contain one or more functions that can be directly started through SOAP or REST service calls. The individual function can receive and return data and is associated with a specific business or data need.

EGL libraries are similar in structure to EGL services in that they contain one or more functions that can receive or return data. However, a library can be started only by some other EGL logic and cannot be exposed as a service. They are typically used to perform common utility functions.

The structure of each of these is simple. Figure 11-6 shows an example of an EGL program and its basic structure.

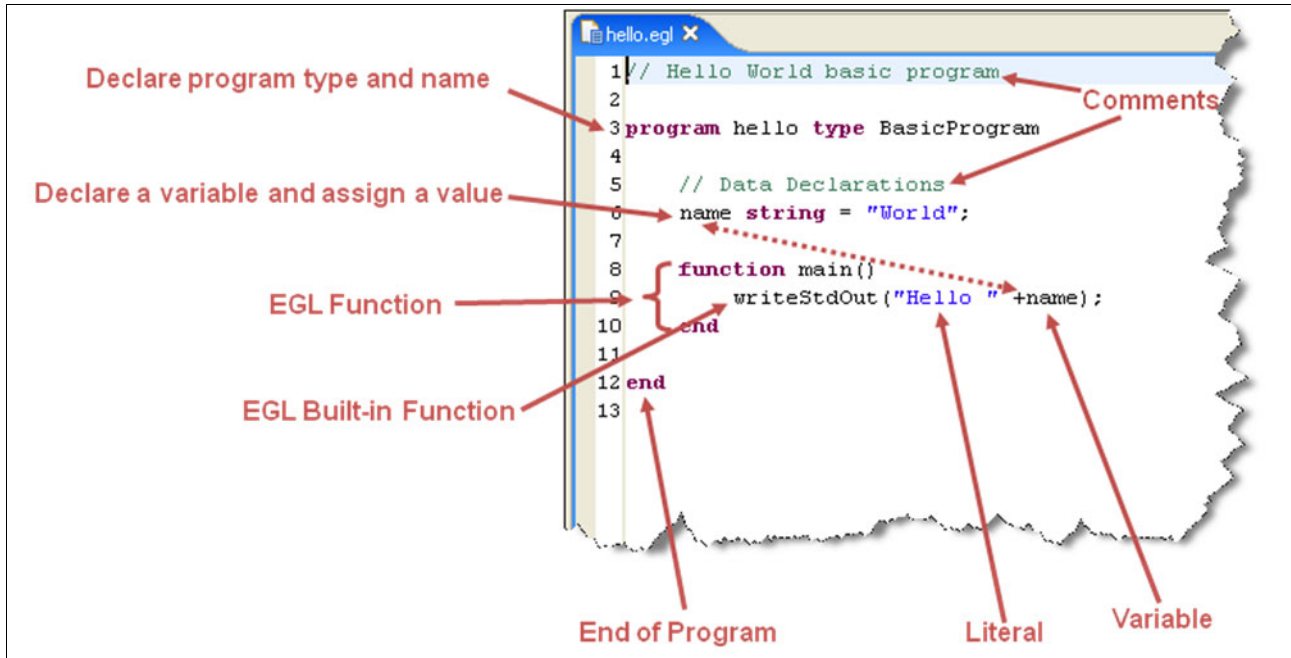


Figure 11-6 Hello World in EGL (example program)

EGL services and EGL libraries share a simple structure and differ only in the keyword that identifies what they are (service versus library). As shown in Figure 11-7, additional logic is added in the function `AddFlights` to insert a record (row) to a SQL table using the “add” verb.

```

SERVICE FlightsService

use ConditionHandlingLib;

Function AddFlights(newRecord Flights, status StatusRec)
// invoke local function to verify data values
if ( IsValid(newRecord) )
// attempt to execute the following code, but catch failures onException
try
add newRecord;
HandleSuccess(status);
onException (exception SQLException)
HandleException(status, exception);
end
else
HandleInvalidDBRecord(status);
end
end

```

Figure 11-7 Basic EGL service syntax

An EGL Interface is similar to a service but it does not hold logic. Instead, the interface defines the records and function names for the programmer to use to start a service (EGL or non-EGL), as shown in Figure 11-8 on page 497.

```

Interface IFlightsService {@xml {name = "FlightsService",
                                namespace = "http://access.as4245rtpraleighibmcom"}}
    Function AddFlights(newRecord Flights, status StatusRec);

    Function AddFlightsList(newRecordList Flights[], status StatusRec);

    Function GetFlights(returnRecord Flights inout, status StatusRec);

```

Figure 11-8 Basic EGL interface example

The syntax to start a service directly or through an interface is the same: "servicename.function(param)". The syntax is shown in Figure 11-12 on page 498.

The external type HostProgram is a special type of interface and service because when it starts the service, EGL implements a remote call to the native RPG program directly. For example, given the RPG source that is shown in Figure 11-9, an EGL services wizard can be run to create the corresponding HostProgram and interface, as shown in Figure 11-10 on page 498 and Figure 11-11 on page 498.

```

000100  F*****
000200  Hnomain
000300  F*  PROGRAM NAME - RPGCLSP1
000400  F*  DESCRIPTION - RPG sample program to calculate mortgage
000500  F*                   used in EGL RUI demo to show RPG invocation
000600  F*****
000700  F*  Data is passed in multiple parameters to program
000800  F*****
000900  F*
001000  F*
001100  F*****
001200  D*
001300  D* main program procedure interface
001400  D
001500  Damortize          PR
001600  DAmt                10p 4
001700  DintRate            10p 8
001800  Dtyears              10i 0
001900  Dreturnval          12p 2
002000  *
002100  Pamortize          B          EXPORT
002200  D                  PI
002300  DAmt                10p 4
002400  DintRate            10p 8
002500  Dtyears              10i 0
002600  dreturnval          12p 2
002700  drate                s          10p 8
002800  dterm                s          10i 0
002900
003000  /free
003100      rate = (1 + intRate / 1200) ;
003200      term = tYears * 12 ;
003300

```

Figure 11-9 RPG source to calculate mortgage information

Figure 11-10 shows an EGL HostProgram defining how to start the RPG program.

```
ExternalType RPGMORTDBG type HostProgram {platformData=[@i5OSProgram{ programName="RPGMORTDBG",
                                                                    programType=NATIVE, isServiceProgram=false, libraryName="*LIBL"}]}

    static function AMORTIZE(AMT decimal(10,4) inout, INTRATE decimal(10,8) inout, TYEARS int inout,
                            RETURNVAL decimal(12,2) inout) { hostName="AMORTIZE"};

end
```

Figure 11-10 EGL HostProgram defining how to start the RPG program

Figure 11-11 shows the interface that is used to start the host program and define the parameters.

```
interface IRPGMORTDBG {@xml{name="RPGMORTDBG" }}
    function AMORTIZE( AMT decimal( 10, 4 ) inout, INTRATE decimal( 10, 8 ) inout,
                      TYEARS int inout, RETURNVAL decimal( 12, 2 ) inout )
                      {@xml{name="AMORTIZE"}};

end
```

Figure 11-11 Interface that is used to start the host program and define the parameters

From a coding perspective, it is easy to start these services from some other logic part. The same is true if you want to use a remote call, as shown in Figure 11-12.

```
// invokes the AddFlights function in flightsService
flightsService.AddFlights(flights, status);

//Invokes RPGMORTDBG RPG program on IBM i as a service call
iRPGMORTDBG.AMORTIZE(AMT, INTRATE, TYEARS, RETURNVAL);

// does a remote call directly to the same RPG program
call "RPGMORTDBG" (AMT,INTRATE, TYEARS, RETURNVAL);
```

Figure 11-12 Example of EGL code to start a service HostProgram or Call statement

As with most languages, you also need a way to identify and work with data. In EGL, this way is called a *record*. Records have associated types, and based on the type, EGL knows what type of processing the programmer wants done and creates appropriate logic and context. For example, as shown in Figure 11-13 on page 499, one type is “sqlRecord” or “SQLRecord” (EGL is case insensitive). The record denotes what table it is associated with and the primary key or keys. This tells EGL to create SQL operations against the named table or tables if this record is referred to in a program.

For SQL, the record also contains fields that are associated with each column. Again, based on this record layout, EGL creates SQL statements that work with the defined columns. EGL creates default statements that are based on the record, but the programmer has full editing ability over what explicit SQL they want to code and run.

The records can be created with wizards that access the associated table and build the field list and primary key information. Figure 11-13 on page 499 shows some examples of “sqlRecords”.


```

record AGENTS type SQLRecord
  {tableNames = [{"FLIGHT400.AGENTS"}], fieldsMatchColumns = yes}

  10 AGENT_NO int      {column="AGENT_NO", isSqlNullable=yes};
  10 AGENT_NAME char(64) {column="AGENT_NAME", isSqlNullable=yes, sqlVariableLen=yes};
  10 AGENT00001 char(64) {column="AGENT00001", isSqlNullable=yes, sqlVariableLen=yes};

end

record Flights type sqlRecord {
  tablenames=[{"FLIGHTS"}],
  fieldsMatchColumns = yes,
  keyItems=[FLIGHT_NUMBER]
}

FLIGHT_NUMBER FLIGHT_NUMBER {column="FLIGHT_NUMBER"};
DEPARTURE_INITIALS DEPARTURE_INITIALS {column="DEPARTURE_INITIALS", sqlVariableLen=yes, maxLen=16, isSqlNullable=yes};
DEPARTURE DEPARTURE {column="DEPARTURE", sqlVariableLen=yes, maxLen=16, isSqlNullable=yes};
DAY_OF_WEEK DAY_OF_WEEK {column="DAY_OF_WEEK", sqlVariableLen=yes, maxLen=16, isSqlNullable=yes};
ARRIVAL_INITIALS ARRIVAL_INITIALS {column="ARRIVAL_INITIALS", sqlVariableLen=yes, maxLen=16, isSqlNullable=yes};

```

Figure 11-13 Examples of an EGL SQLRecord

There are several input and output operations in the EGL language. The EGL language element that is used to retrieve one or more rows is the “get” statement. This statement is automatically converted into the necessary EXEC SQL statements to implement the requested operation. You, as the programmer, do not need to be concerned with this. You need to code only the SQL statement if the default is not what you want. The default statement that is used on a “get Flights” statement is shown in Figure 11-14. Earlier, in Figure 11-7 on page 496, there is an example of the “add” for a SqlRecord named “searchRecord” passed to the service. The “add” inserts one row in the table.

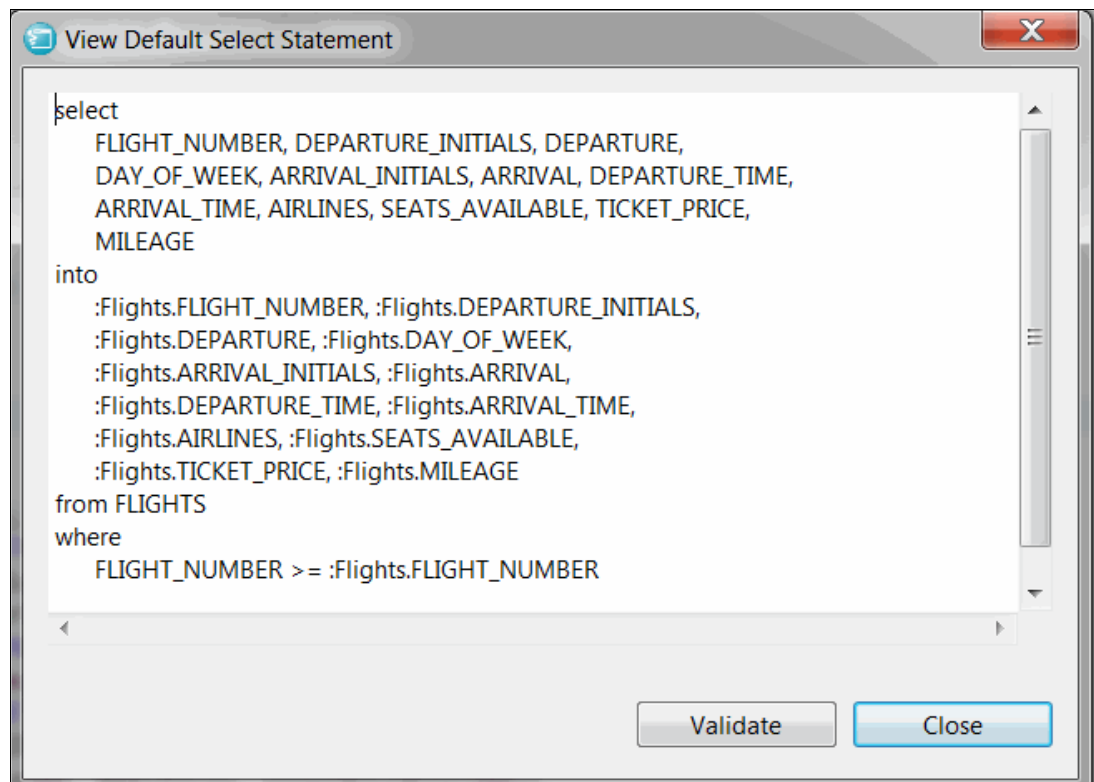


Figure 11-14 Default SQL select statement for the “Flights” table

There are other record types, such as indexedRecord, serialRecord, and mqRecord. The concepts for these types and the verbs to read and write records to files or WebSphere MQ queues are the same as for sqlRecords. What changes are the properties that are defined on the record, such as file name or queue name that are unique to that type of record. The “get” and the “add” are used and based on the record type. EGL then creates the correct implementation to read and write files or get and put for an WebSphere MQ queue.

This is only a brief look at the syntax or language elements that are used to build business and data services. EGL has much more capability that is related to statements, operations, built-in functions, data types, error handling, and other syntax that is used to build applications. The *EGL Reference Guide* and *EGL Programmers Guide* in the Rational Business Developer V9.0 information center contain more information about the complete EGL language and tools:

<http://pic.dhe.ibm.com/infocenter/rbdhelp/v9r0m0/index.jsp>

11.4 EGL rich user interface for web and mobile interfaces

One of the aspects of application modernization is creating a modern interface to the applications. Today, this means both an interface that is usable on the web and from mobile devices, similar to the windows that are shown in Figure 11-15. For this aspect, EGL provides a RUI as part of its language syntax to let you build sophisticated pages and applications that can be deployed on the web or through mobile devices. The RUI technology can access the EGL services or non-EGL services to provide access to existing IBM i databases or to reuse existing RPG programs.

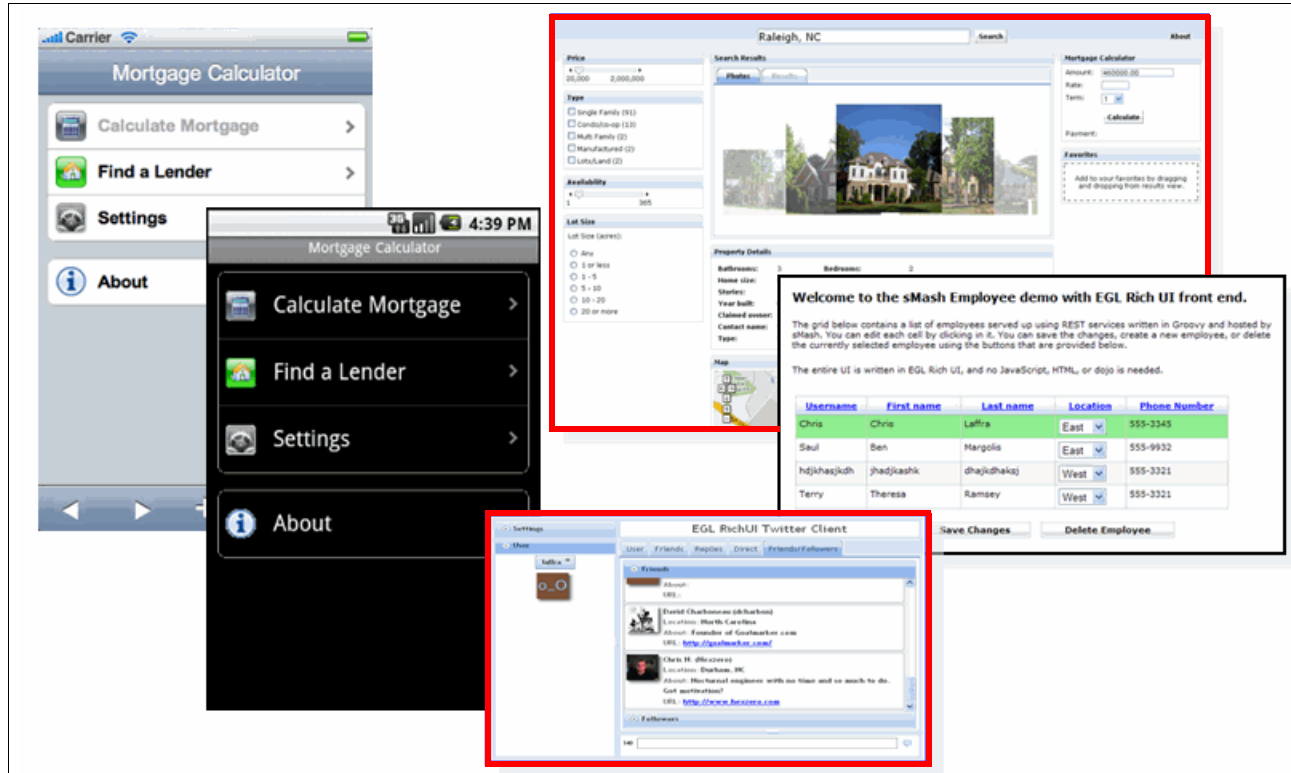


Figure 11-15 EGL Web 2.0 and mobile interface for today's business applications

First, consider the current requirements and challenges in building today's web applications and then how EGL can provide a solution.

11.4.1 The need for Web 2.0 and rich internet applications

Web applications are no longer static, server generated collections of pages. Web 2.0 and rich internet applications (RIAs) represent the current technology for web applications because of the following reasons:

- ▶ Provides capabilities of a desktop application, but with the manageability of a web application
- ▶ Enabled by technology such as JavaScript and Ajax
- ▶ Lightweight and built on open standards

RIAs also provide a richer user experience compared to traditional Web 1.0 applications (implemented with JSP technology). Here are the reasons:

- ▶ The overall interface is simplified, yet more powerful for the user
- ▶ Enables mashups, which is the ability to put together related data from multiple sources
- ▶ Implements more processing on the client (for example, validation) in the browser
- ▶ Implements what business users are accustomed to within their everyday personal web experience

The net effect is that Web 2.0 provides the required vehicle for delivering richer, more powerful business applications.

11.4.2 Challenges in building Web 2.0 and rich internet applications

There are many challenges that are associated with building these Web 2.0 and RIAs. Here are some of the challenges:

- ▶ This type of development is usually the domain of “tech heads” versus business programmers.
- ▶ The developer must learn multiple complex technologies (JavaScript, Ajax, JSON, SOAP, and others).
- ▶ Compounds the skill and tool silos, which results in fragmentation of staff, which increases the difficulty of retraining the current business programming staff.
- ▶ Most tool solutions are either front-end or back-end focused, but not both, which results in code duplication and manual efforts to keep code in sync.
- ▶ Many solutions are built on Web 1.0 style architectures, which are not an ideal programming model for building RIAs.

These challenges combine so that building RIAs needing IBM i access require much time, many different tools, many different languages, lengthy retraining, and difficult collaboration because of fragmented teams and skills.

11.4.3 Web 2.0 and rich internet applications with EGL

EGL includes the RUI in its language and tools to enable programmers to quickly build Web 2.0 and RIAs while modernizing existing applications. The way this is accomplished is as follows:

- ▶ Using the same single language to deliver Web 2.0 based applications and to build the business and data logic (end-to-end).
- ▶ Providing JavaScript based widgets for business-oriented applications (Dojo and EGL-provided), including date and time pickers, sliders, graphs, currency, and other input widgets (with built-in validation), tab and accordion containers, tree, and others.
- ▶ Compiles the EGL source into standard JavaScript and Ajax:
 - No browser plug-ins are required.
 - Works with all major browsers.
- ▶ Easing learning requirements for ALL types of developers. Uses familiar EGL syntax to use and work with Dojo widgets in your applications.
- ▶ Using or integrating with popular and open existing JavaScript libraries, like Dojo, Ext JS, and jQuery and style sheets (CSS) by using Eclipse-based development, testing, and debugging, including:
 - A powerful visual editor to build your pages through dragging.
 - Real-time running and debugging of the RUI source.
 - Wizards to enable creation of records to exchange information as XML or JSON.
 - Automatic creation of widgets based on existing record definitions.
- ▶ Consumption of all types of web services, including multiple simultaneous asynchronous invocations.
- ▶ Support for standard Model View Controller (MVC) architectures.

All of these items combined provide greater productivity for your programmers and let you produce high-quality web applications for your users and customers.

11.4.4 EGL rich user interface syntax and tools

EGL continues to use the same language constructs for EGL RUI applications as used in the business logic that is described in 11.3, “A brief look at the EGL language” on page 495. The definition for a page is coded in a “RUIHandler” and the handler contains a combination of source to define the layout and widgets on the page, validation logic, navigation logic, and invocations of back-end services to handle the database interactions.

To aid productivity, when you create a “RUIHandler”, the EGL RUI visual editor opens and you can design the layout using a predefined grid and define variables to populate the page. There is a palette that contains all the widget types that are supplied by EGL or Dojo. This includes widgets for standard web applications and mobile applications.

In Figure 11-16 on page 503, you see the editor with three tabs at the bottom: Design, Source, and Preview. The Design tab is where you can lay out the design and define variables for your page. You also see the palette, from which you can drag different widgets as needed to lay out on the page. There are various drawers in the palette with different widget types. They are not shown here, but there are also mobile widgets to be used inside of a browser on a mobile device.

A grid is used by default to help with alignment. In Figure 11-16, five widgets have been created: two labels, two text fields to enter and display data, and a button to click. When the button is clicked, an event is triggered and you are able to name an EGL function to start.

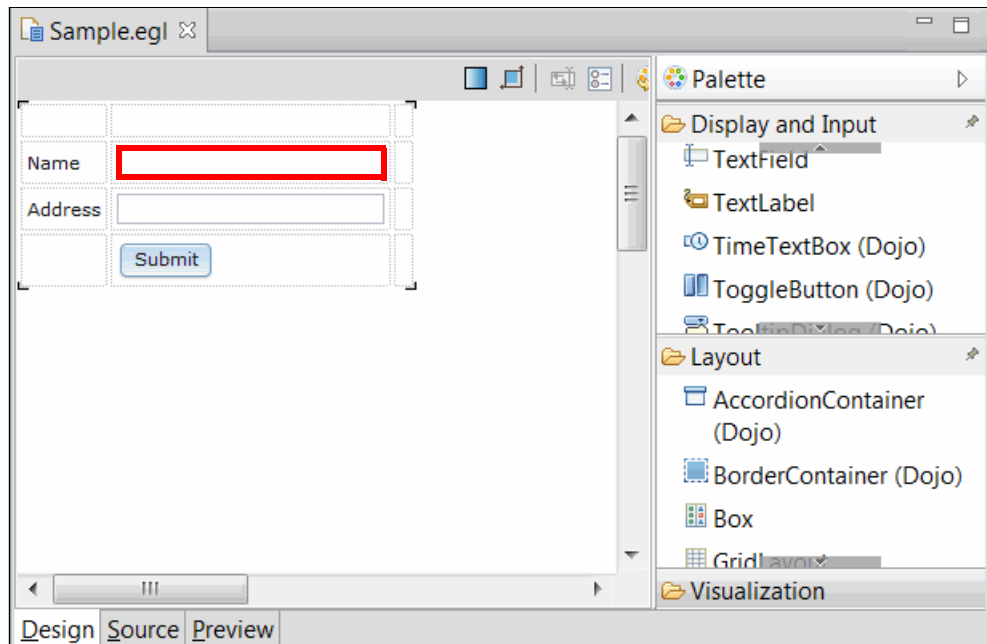


Figure 11-16 EGL RUI visual editor

Figure 11-17 shows the source that is created after dragging the widgets on to the grid. The source is shown in the editor by clicking the **Source** tab. The variables and widget types are placed in the source code within the GridLayout at the top of the source.

The functions hold the simple logic to start a service when the button is clicked and then to receive the data and populate the page when the service returns the results. For more information, see the comments.

```

handler Sample type RUIhandler {initialUI = [ ui ],onConstructionFunction = start, cssFile="css/SampleRUIProject.css", title="Sample"}

// Variables defining layout and content of the page.
ui GridLayout{ columns = 3, rows = 4, cellPadding = 4, children = [ AddressLbl, Address, SubmitBtn, NameLbl, Name ]};
Name DojoTextField{ layoutData = new GridLayoutData{ row = 2, column = 2 }};
NameLbl TextLabel{ layoutData = new GridLayoutData{ row = 2, column = 1 }, text = "Name" };
SubmitBtn DojoButton{ layoutData = new GridLayoutData{ row = 4, column = 2 }, text = "Submit", onClick ::= SubmitBtn_onClick };
Address DojoTextField{ layoutData = new GridLayoutData{ row = 3, column = 2 }};
AddressLbl TextLabel{ layoutData = new GridLayoutData{ row = 3, column = 1 }, text = "Address" };

// Declaration of service to call
mysvc mysvc {@bindService};

function start()
end

// Function invoked when button is clicked
function SubmitBtn_onClick(event Event in)

    //invokes service passing the name entered but does not wait for return
    call mysvc.returnAddress(name.value) returning to myfunction;
end

// Function invoked after return from service
// specified in "returning to" on call statement
// retResult would be the address returned by the service
function myfunction(name string in, retResult string in)
    address.value = retResult;
end
end

```

Figure 11-17 Sample EGL RUI handler source code (see comments in code for details)

The Preview tab is shown in Figure 11-18. Switching to the Preview tab enables two items:

- ▶ It runs the page and lets you see what the page looks like when presented to the user.
- ▶ It allows you to run the page, including starting events, services, and running validation and other logic so that you can test the entire page without having to deploy it to the server first.

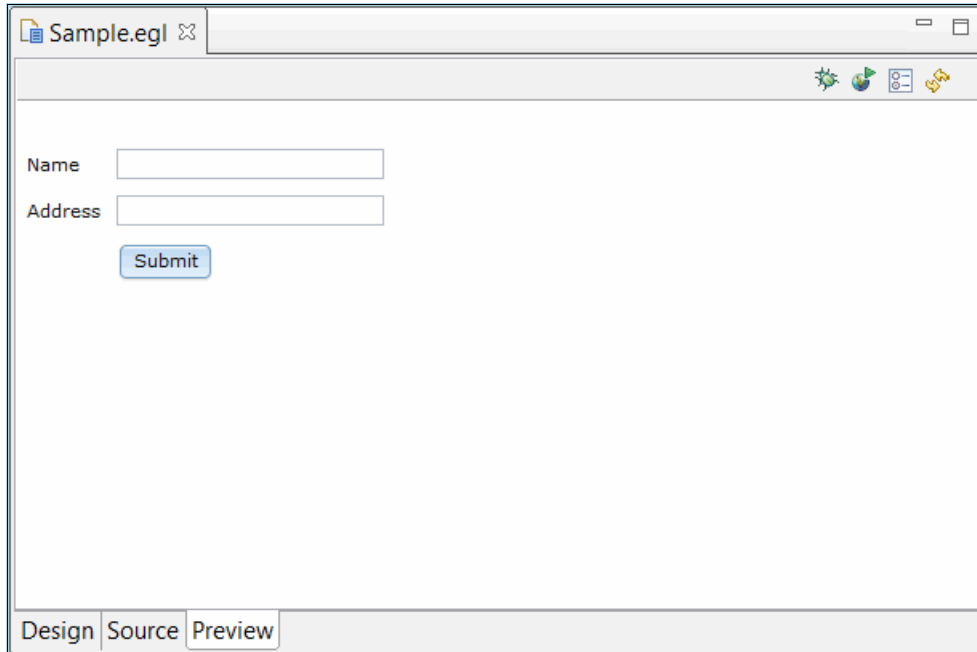


Figure 11-18 Sample EGL RUI in the Preview tab

An additional capability within the EGL visual editor is the ability to drag records on to a page. When this is done, the editor automatically determines the best widget, based on the data types within the record. For example, look at Figure 11-19 on page 505, where an EGL record named "Order" is dragged on to the page. "Order" contains several header fields that are related to the order and then an array of lines that represent the individual items that are contained in the order. The "Insert Data" dialog box opens and shows the fields with the header fields being individual widgets based on their data type and the array becoming a "DataGrid" with multiple columns.

The labels, widget type, and the widget name (variable name) can all be changed in this dialog box. The wizard uses an MVC pattern to automatically bind UI elements to the underlying data.

The records that are used as input can represent database tables, external RPG input/output data, or SOAP or REST service input/output data.

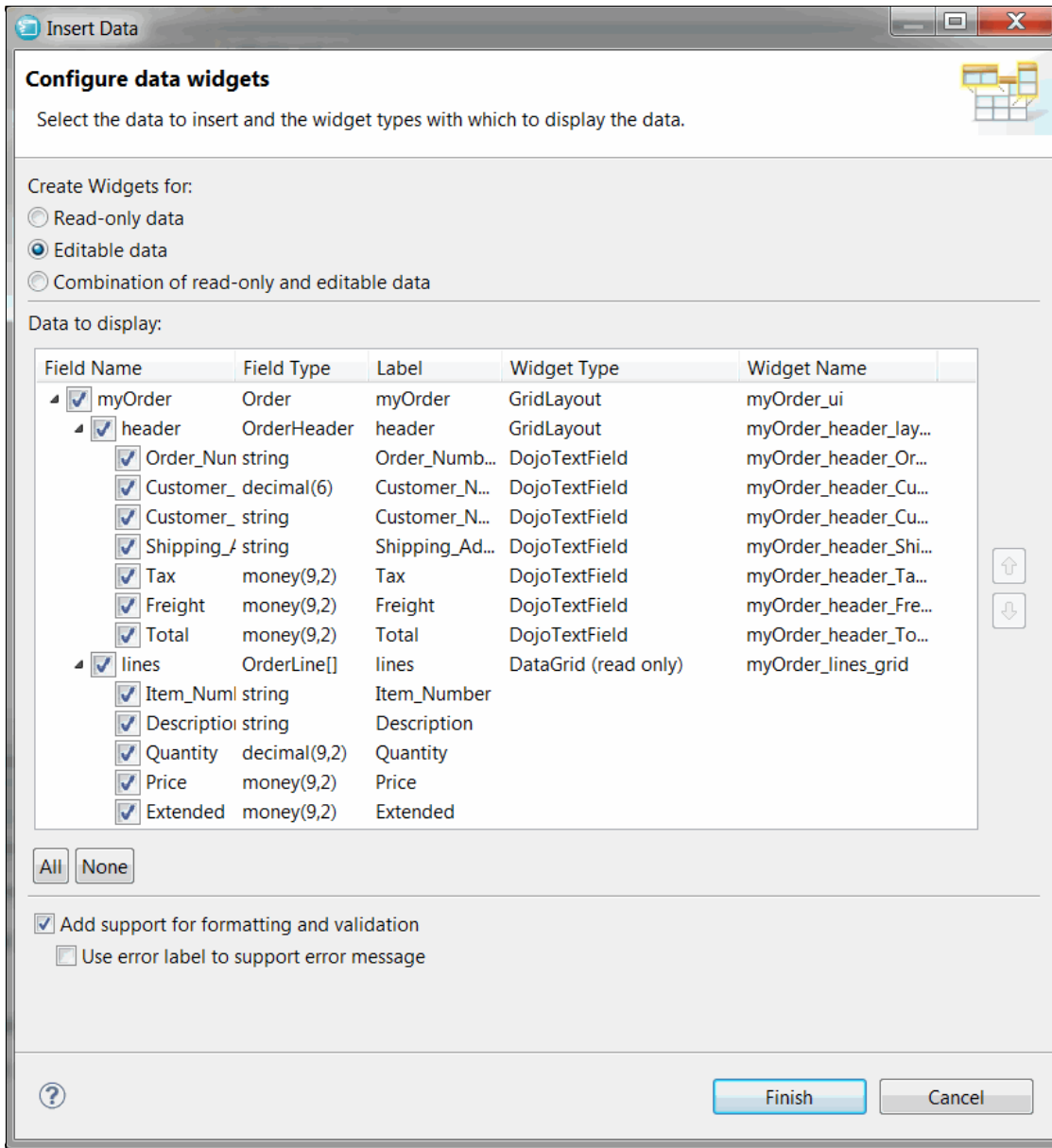


Figure 11-19 EGL insert data wizard

The resulting page, as shown in the design view for this page, is shown in Figure 11-20. As you can see, much of the design work is done for you with existing records that were created when you built a data access application or used the HostProgram wizard to access an RPG program.

The screenshot shows a web form layout. At the top, there are seven input fields, each with a label to its left: Order_Number, Customer_Number, Customer_Name, Shipping_Addr, Tax, Freight, and Total. The Tax, Freight, and Total fields contain the value "\$0.00". Below these fields is a table with five columns: Item_Number, Description, Quantity, Price, and Extended. The table has a header row and several empty rows below it. The word "lines" is written vertically on the left side of the table area.

Figure 11-20 EGL RUI layout that is created by dropping a record on a page in the visual editor

Just like all other forms of code in EGL, the RUI handler can be debugged at the EGL source level. This allows you to step through the application at the EGL source level, view and change variables, and even alter the execution path if wanted. Combined with the ability to debug the services that are written in EGL, you can do “end-to-end” debugging of your web applications.

For more information about building EGL RUI applications, including links to tutorials, blog entries, and other useful information, see the following links:

- ▶ Rational Business Developer V9.0 information center:
<http://pic.dhe.ibm.com/infocenter/rbdhelp/v9r0m0/index.jsp>
- ▶ Rational EGL Community Edition:
<https://www.ibm.com/developerworks/downloads/r/eglcommunity/>
- ▶ EGL Cafe:
<https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=3e2b35ae-d3b1-4008-adee-2b31d4be5c92>

11.5 EGL: From the database to the web

EGL provides modernization capabilities to allow you to reuse existing SQL tables on IBM i through the easy creation of business logic and Web 2.0-based (JavaScript) web interfaces to present and work with that data.

This section describes how the EGL Data Access Application wizard is used to create a functioning web service, which provides many functions to access the database on IBM i. These service functions can be used by many applications and clients. Also, this section describes how the service can be used by an EGL RUI to display the results of the database access on a web page.

11.5.1 Creating a data access application and services

A data access application is EGL logic that provides fully functioning Create, Read, Update, and Delete operations against your IBM i database. The logic represents a set of functions that is accessible as a SOAP-based or REST-based web service.

To follow this tutorial, you must start either Rational Business Developer or Rational Developer for i V9.0, RPG and COBOL + Modernization Tools, EGL edition.

11.5.2 Creating a data access application

Right-click in the Project Explorer view and select **New** → **Other...**, or click **File** → **New** → **Other...**. The window that is shown in Figure 11-21 opens. You can use filter text to minimize the choices.

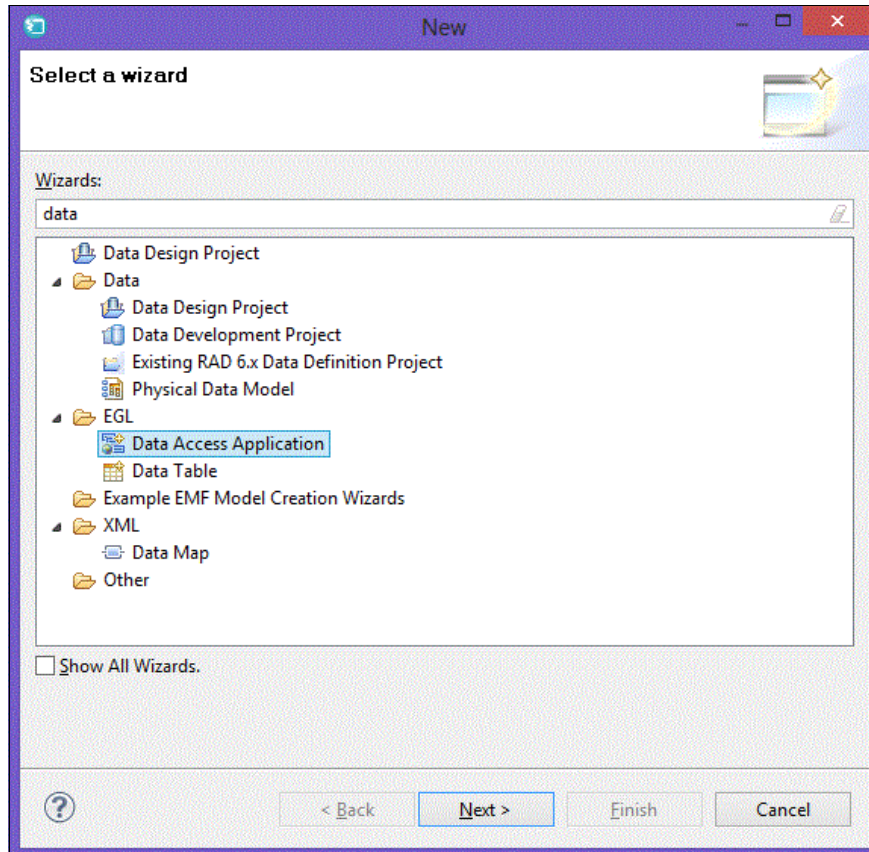


Figure 11-21 Selecting a wizard

Click **Next**, and, in the next window (Figure 11-22), you can name the project. In this example, *backend* is chosen because this project contains back-end services that run on the application server.

To define which database and tables to access, click **New...** to define a database connection to the IBM i machine that contains your library and tables.

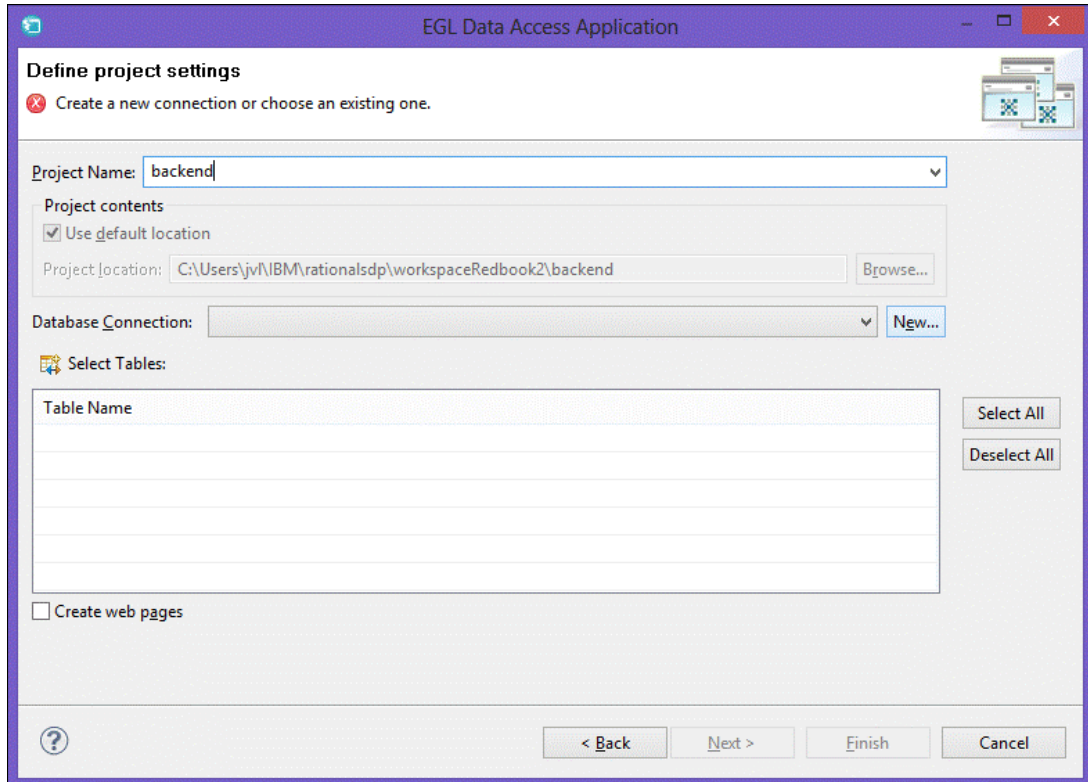


Figure 11-22 Defining project settings in the EGL Data Access Application wizard

In the New Connection window (Figure 11-23), you create a connection to the database on your IBM i.

The first window already has the properties that are needed to connect to the database on your IBM i. The default JDBC driver that is used to access the IBM i is provided within the Java toolkit (JT400 toolkit). It requires connection information, such as the IBM i host name or IP address and the user ID and password for the IBM i system. The default schema is not required and the connection URL is created for you.

Click **Test Connection** to ensure that the connection can be established.

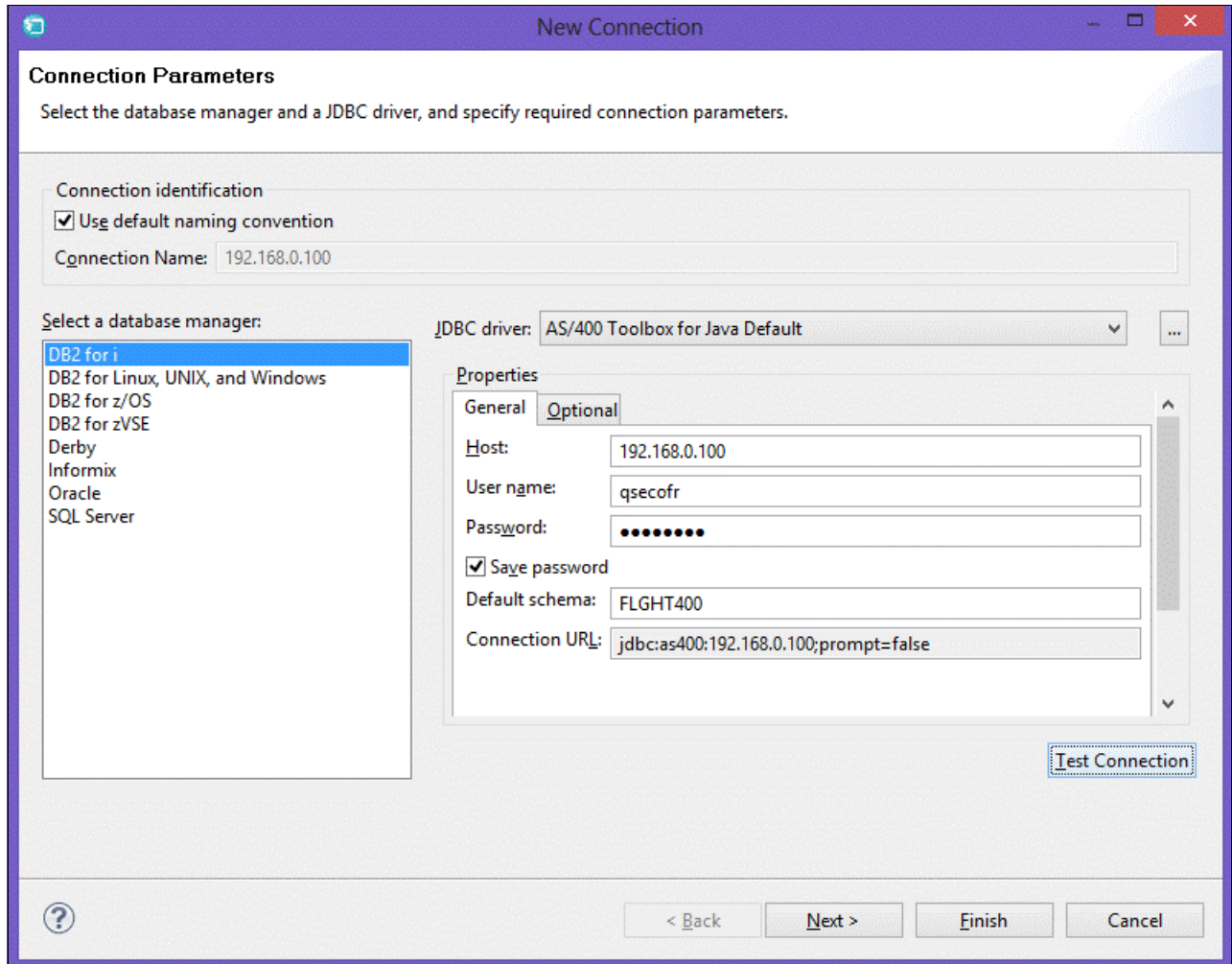


Figure 11-23 The general properties to connect to your IBM i

After clicking **Next**, the window that is shown in Figure 11-24 shows an overview of the libraries to which the user has access. For this example, you can select the library “FLGHT400” because the tables within this library are the ones you want to use, as shown in Figure 11-25 on page 512.

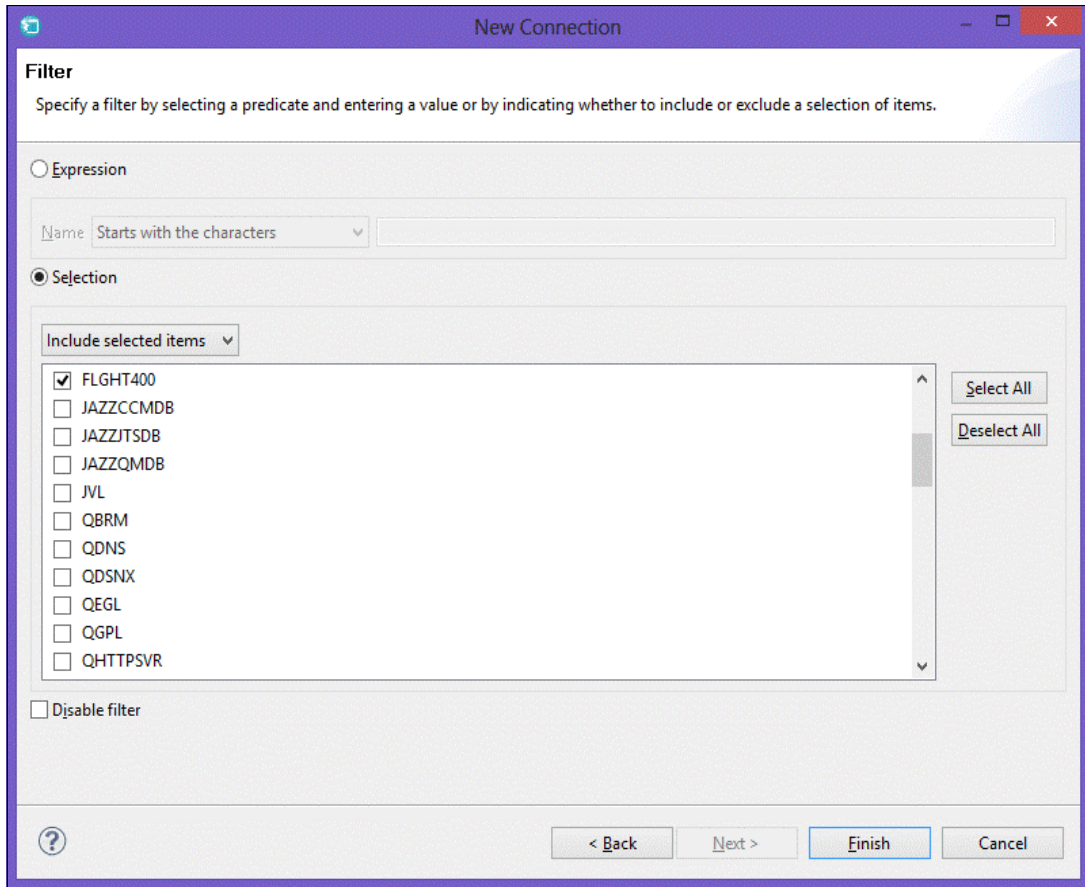


Figure 11-24 Overview of libraries

After clicking **Finish**, as shown in Figure 11-25, you get a list of the tables in the selected library. You can now select the tables that you want to use to create the data access application.

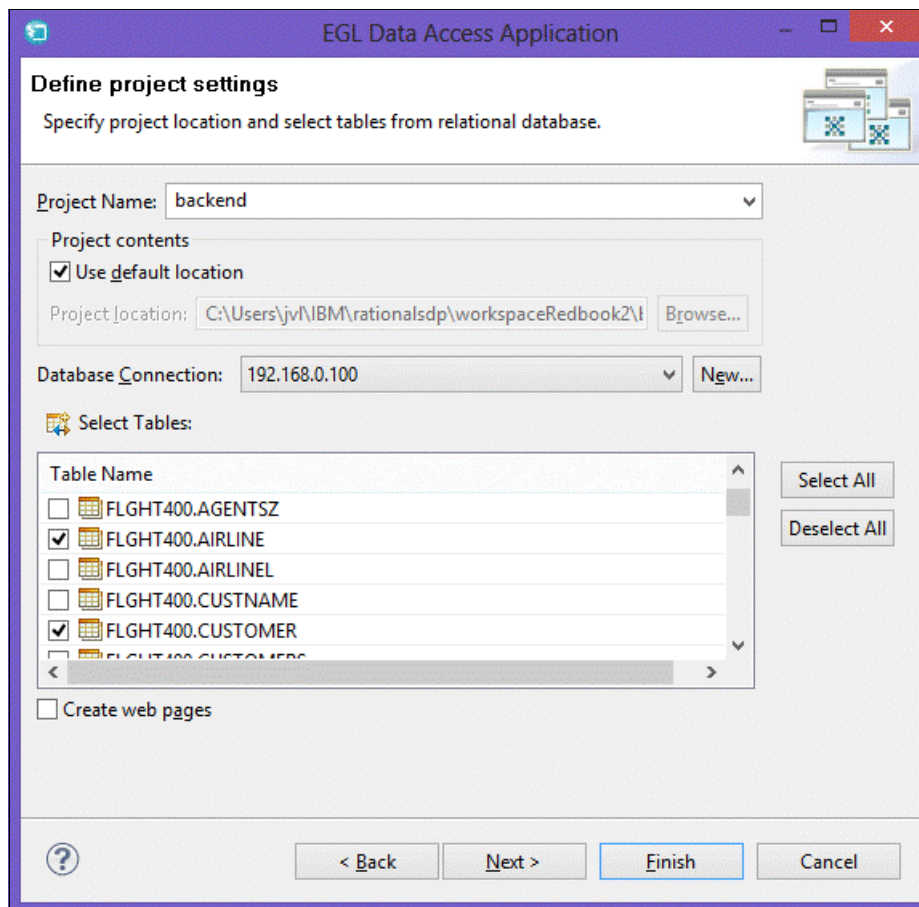


Figure 11-25 Choosing the tables for the data access application

In the window that is shown in Figure 11-26, you get separate tabs for each table. Each tab is a list of columns for the particular table. Here you can review the tables and change information, such as what the primary key or keys should be.

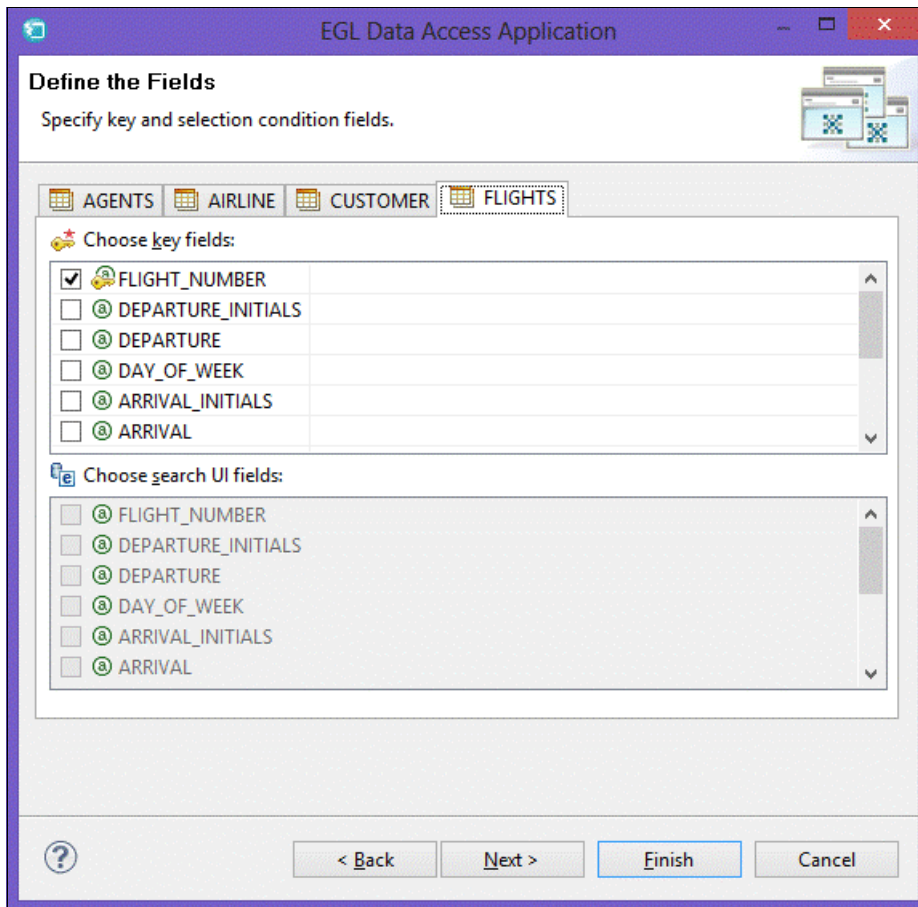


Figure 11-26 Overview of the tables where you can change the key fields

After clicking **Next**, give the package a name, as shown in Figure 11-27. The package is essentially a grouping of similar functions and is used to hold the output from this wizard. In this case, the package contains the logic, which accesses the tables in the “Flight400” library.

You can then choose whether you want to output EGL libraries or EGL services. You should choose EGL services because the result is EGL logic that is usable as a web service.

You can also select the **Qualify table names with schemas** check box to use qualified table names. In this case, the schema (library name) is hardcoded in the created SQL records. If you want to choose the schema at run time or to search for the table using a library list, do not use qualified table names. This allows you to be able to specify the library in your application server or make it dependent on the user profiles on your IBM i.

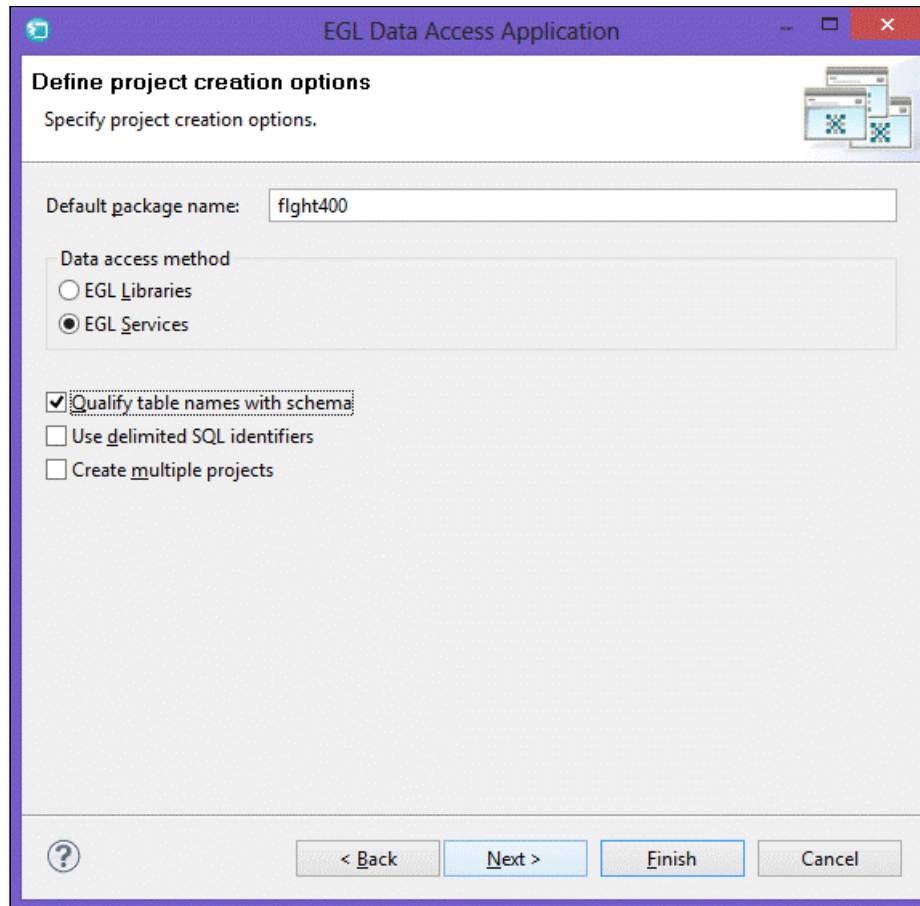


Figure 11-27 Additional options for the data access application

The window that is shown in Figure 11-28 on page 515 opens and shows an overview of everything that you chose. When you click **Finish**, a separate data access service is created for each table.

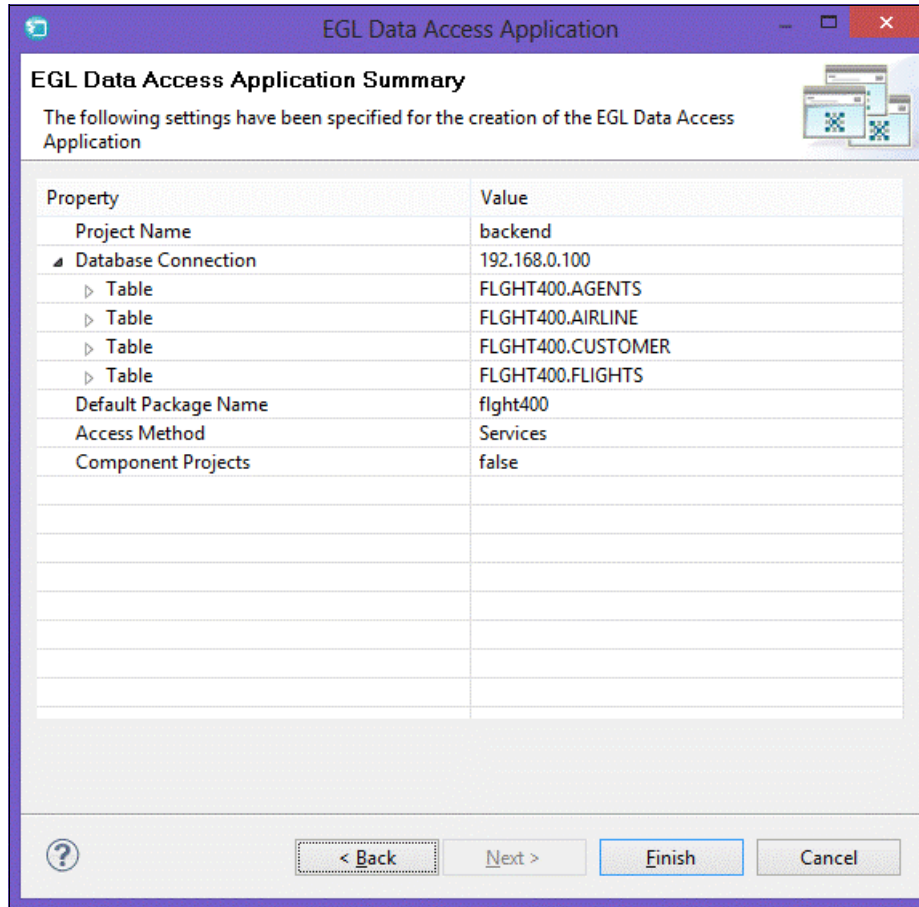


Figure 11-28 Overview of the settings for the EGL data access application

Now the workbench is shown again. You can expand the newly created project and browse the source to see what is created. The functions that are used to access the FLIGHTS table are shown here. There are functions in the services to retrieve a list of all the data of a table, functions that expect a key value and return one row, and other functions. In Figure 11-30 on page 517, you see a list with self-explanatory names. Each of these is called a function in EGL.

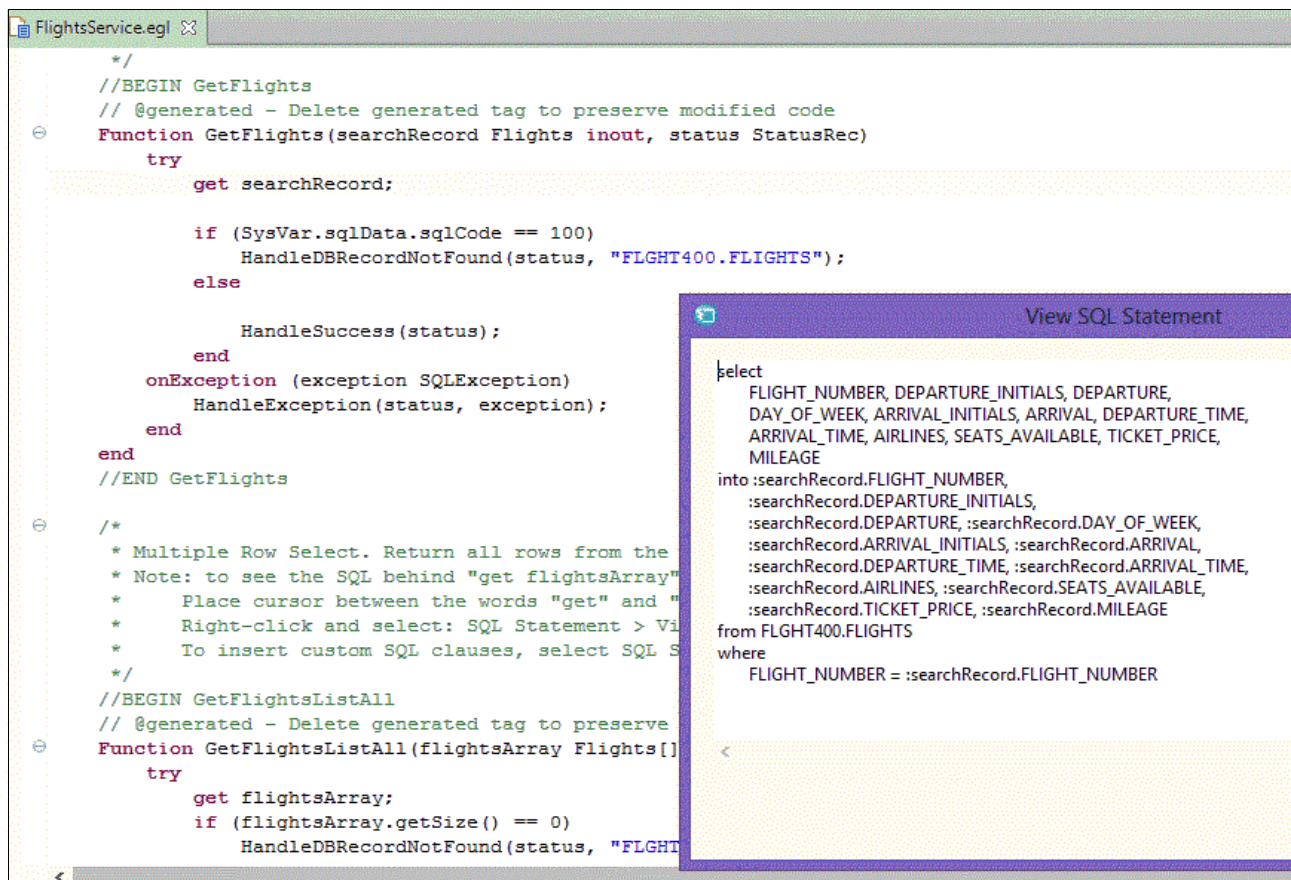
In Figure 11-29, you can see the **GetFlights** function. This function expects two parameters:

- ▶ **searchRecord** contains selection criteria.
- ▶ **statusRec** contains status criteria. This contains information, such as a message or a Boolean to specify whether there are errors.

The **SELECT** statement is run when the **get** statement in the function **GetFlights** is started and the data is returned in the **searchRecord** variable.

EGL uses simple verbs such as “get”, “add”, “update”, “delete”, and “replace” to represent operations on a table or file.

In a function, if you click an implicit SQL statement, you can choose to show the related explicit SQL that is run, as shown in Figure 11-29.



The screenshot shows an IDE window titled "FlightsService.egl". The code defines a function `GetFlights` that takes `searchRecord` and `status` as parameters. Inside the function, there is a `try` block with a `get searchRecord;` statement. Below this, there is an `if` statement checking `SysVar.sqlData.sqlCode == 100`. If true, it calls `HandleDBRecordNotFound` with `status` and `"FLGHT400.FLIGHTS"`. Otherwise, it calls `HandleSuccess` with `status`. There is also an `onException` block for `SQLException` that calls `HandleException`. The function ends with `end` and `//END GetFlights`. Below this, there is a comment block explaining how to view the SQL statement behind the `get` statement. Then, another function `GetFlightsListAll` is defined, which takes a `flightsArray` parameter and has a `try` block with `get flightsArray;` and an `if` statement checking `flightsArray.getSize() == 0`, which calls `HandleDBRecordNotFound` with `status` and `"FLGHT400.FLIGHTS"`. A dialog box titled "View SQL Statement" is open over the code, showing the following SQL query:

```
select
  FLIGHT_NUMBER, DEPARTURE_INITIALS, DEPARTURE,
  DAY_OF_WEEK, ARRIVAL_INITIALS, ARRIVAL, DEPARTURE_TIME,
  ARRIVAL_TIME, AIRLINES, SEATS_AVAILABLE, TICKET_PRICE,
  MILEAGE
into :searchRecord.FLIGHT_NUMBER,
:searchRecord.DEPARTURE_INITIALS,
:searchRecord.DEPARTURE, :searchRecord.DAY_OF_WEEK,
:searchRecord.ARRIVAL_INITIALS, :searchRecord.ARRIVAL,
:searchRecord.DEPARTURE_TIME, :searchRecord.ARRIVAL_TIME,
:searchRecord.AIRLINES, :searchRecord.SEATS_AVAILABLE,
:searchRecord.TICKET_PRICE, :searchRecord.MILEAGE
from FLGHT400.FLIGHTS
where
  FLIGHT_NUMBER = :searchRecord.FLIGHT_NUMBER
```

Figure 11-29 Sample of the code that is created to return one row

You can also look at the functions of the data access application, as shown in Figure 11-30.

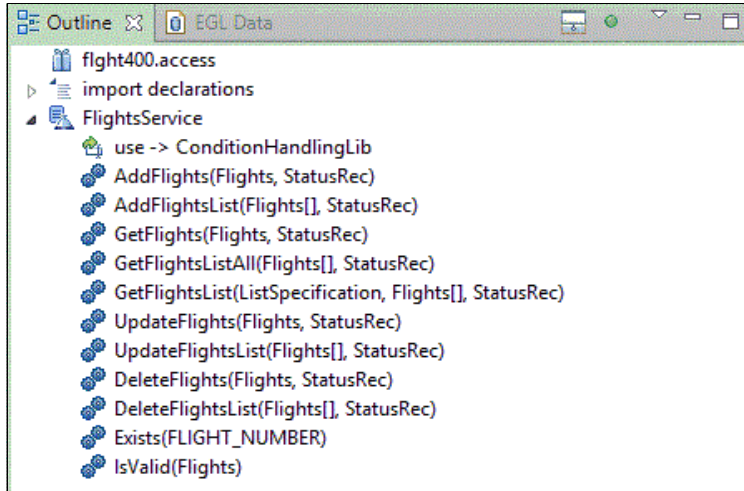


Figure 11-30 Some functions of the data access application

11.5.3 Creating services and testing a local web service

As shown in Figure 11-31, you can generate a WSDL file from the EGL service you created in the previous section. With this WSDL file, your newly created service can be shared with many tools to enable this logic to be started in many web environments and from many other tools. From an EGL web client, the logic can also be started as a REST service.

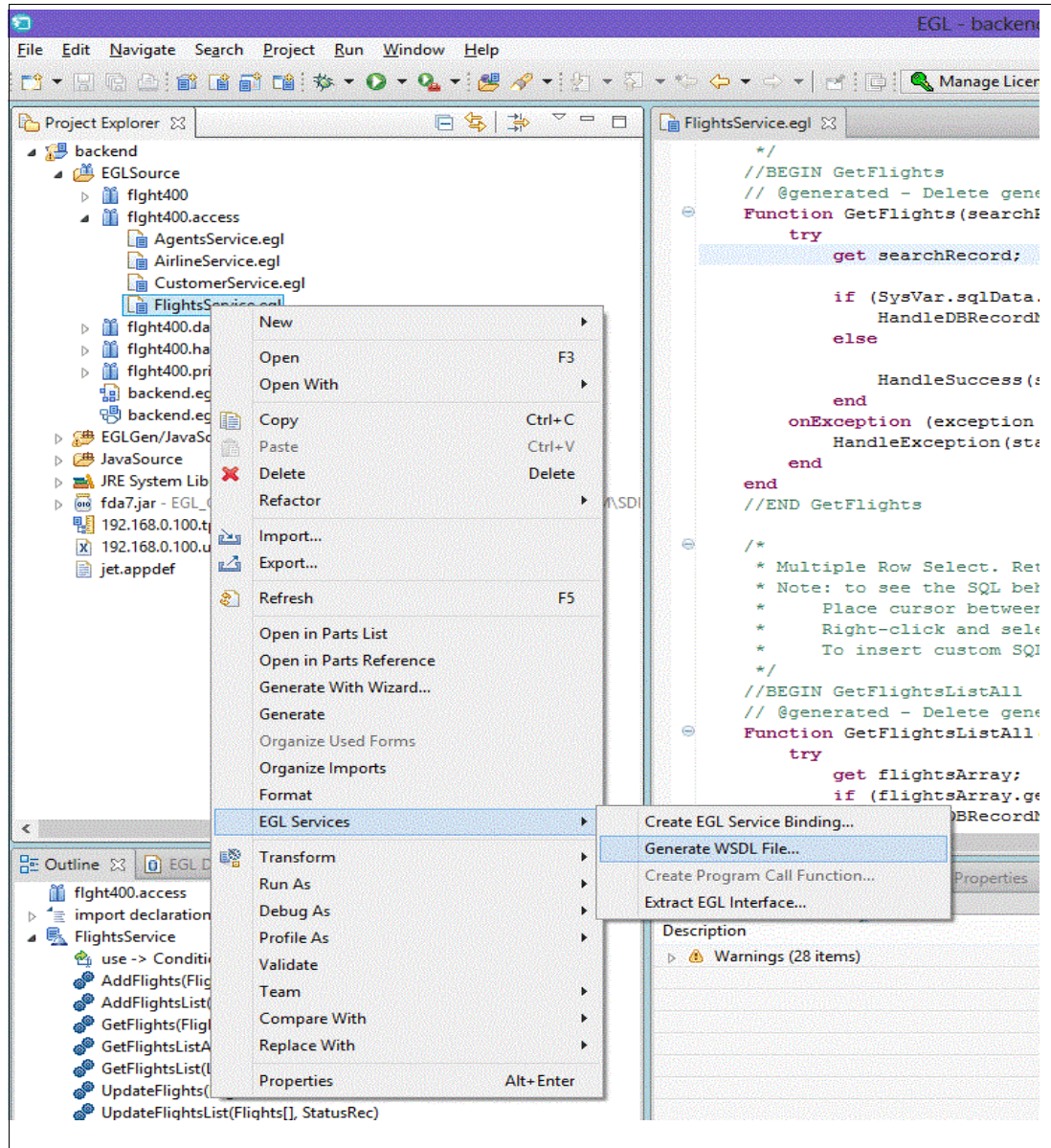


Figure 11-31 Create a WSDL file for a service

You can change some settings (for example, the port number) while creating a WSDL file, as shown Figure 11-32 on page 519. Most people use the defaults that are presented in the wizard.

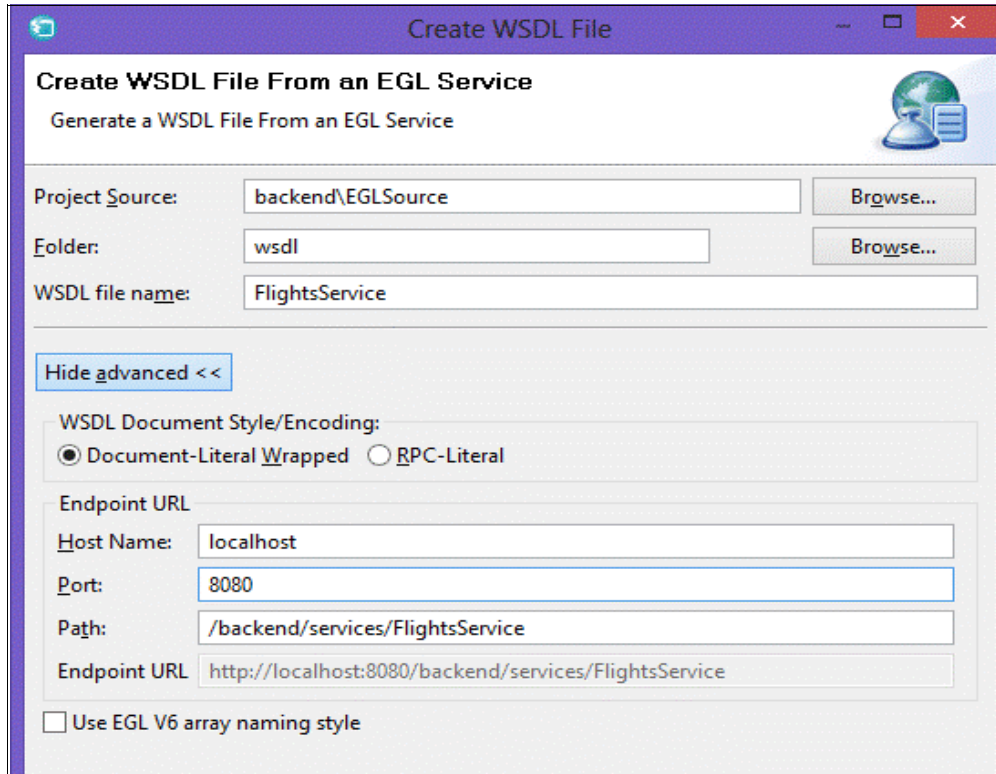


Figure 11-32 Changing the preferences of the WSDL file

If you deploy the service on a Tomcat web server in your workspace, you can test the service function by using the WSDL file and the Web Services Explorer. To do this, you must add a server to run the service on and deploy your project on it.

Start by adding a server to your workspace. To do this, right-click in the **Servers** view and select **New** → **Server**, as shown in Figure 11-33.

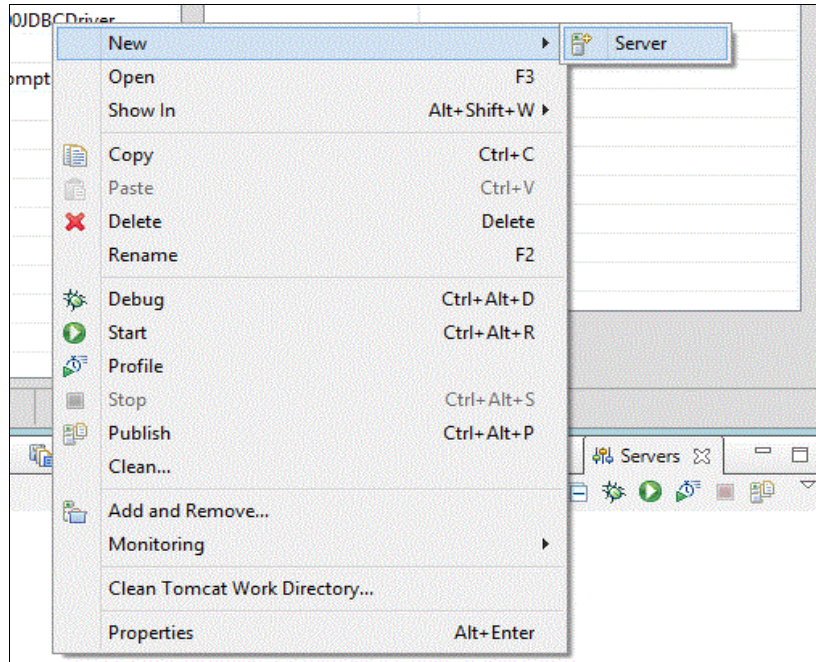


Figure 11-33 New server window

The New Server window opens. Choose a server type and give it a name. If you do not have a server runtime environment on your computer, choose **Add...** to browse to the Tomcat folder (for example) that you downloaded, as shown in Figure 11-34 on page 521 and Figure 11-35 on page 521.

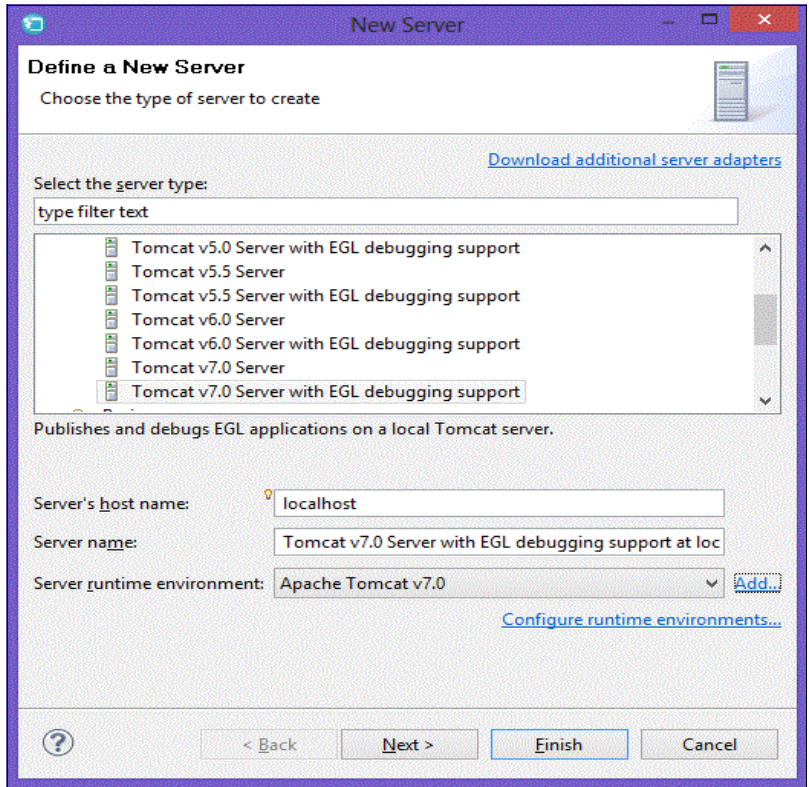


Figure 11-34 Defining a server type

A new server runtime environment window opens, as shown in Figure 11-35.

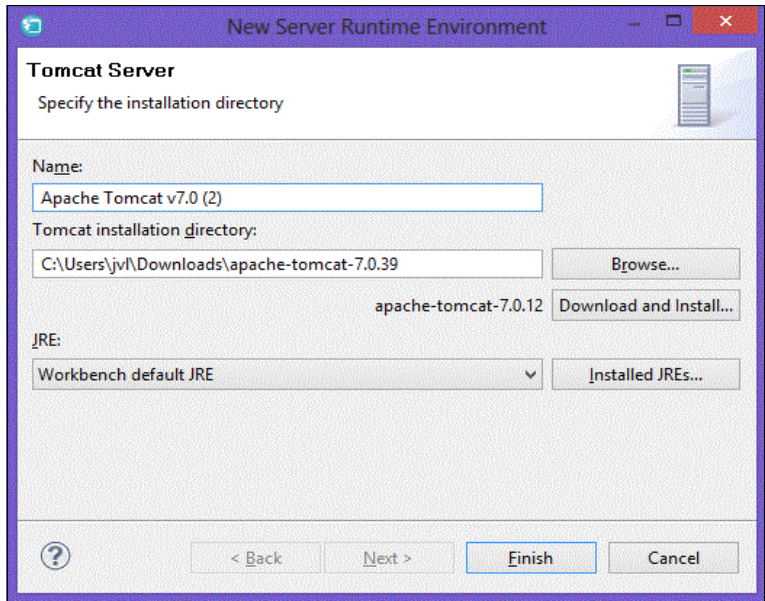


Figure 11-35 Adding a server runtime environment

In the Project Explorer, there is now a servers folder. Open this folder and browse to your server installation. Find the context.xml file. Open this file and add a resource so that Tomcat knows how to access the database, as shown in Example 11-1.

Example 11-1 Resource definition to Tomcat

```
<Resource driverClassName="com.ibm.as400.access.AS400JDBCdriver"
name="jdbc/SAMPLE" password="password" type="javax.sql.DataSource"
url="jdbc:as400:192.168.0.100;prompt=false" username="user"/>
```

In Figure 11-36, you see an example of what the context.xml file looks like.

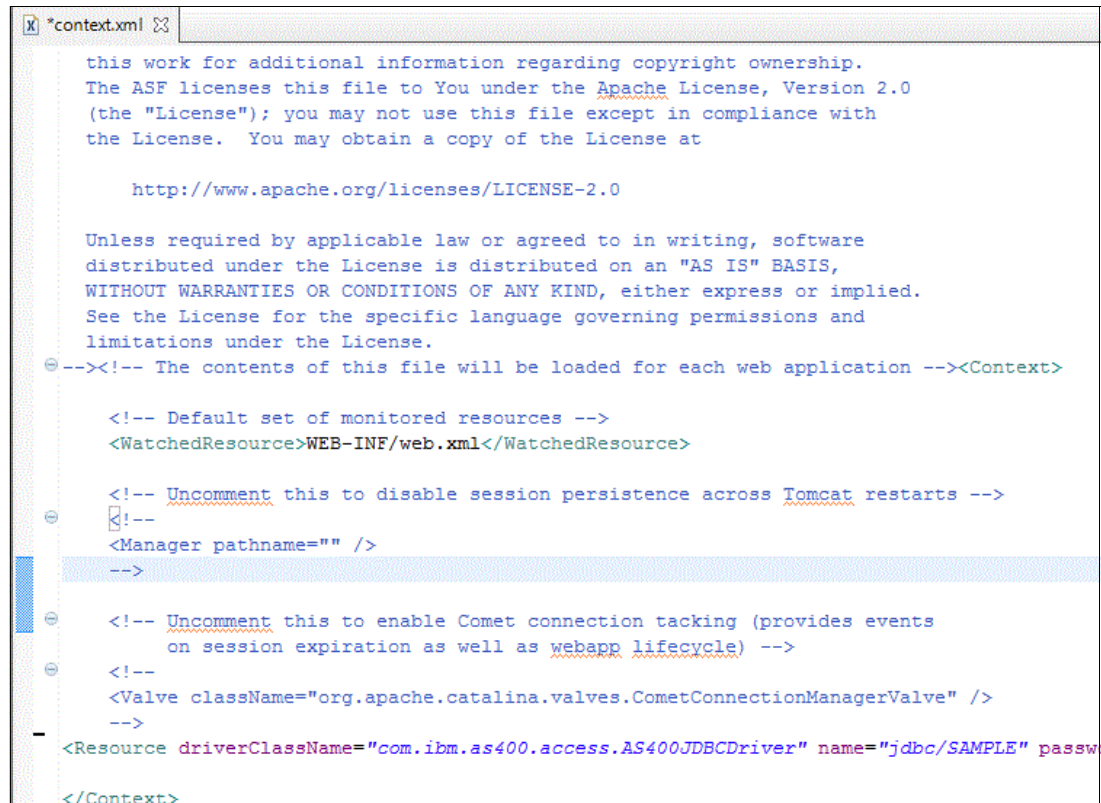


Figure 11-36 The context.xml file for your Tomcat server

Your Tomcat server can now access the database on your IBM i. To deploy the project on your newly created server, right-click the project and select **Deploy EGL Project**. Then, right-click the server and select **Add and Remove...** (see Figure 11-37 on page 523).

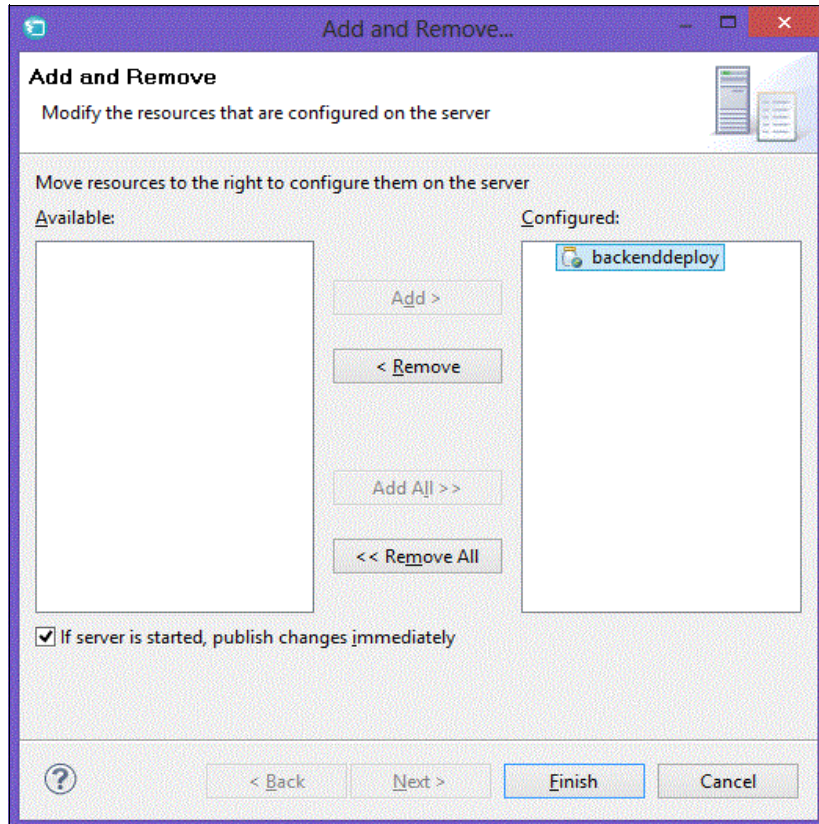


Figure 11-37 Choosing projects to add on your server

In this window, select the available project and click **Add** to place the project in to the configured projects column. Click **Finish**. Now, right-click the server and click **Start**.

Now that the service is deployed, right-click the WSDL file, click **Web Services**, and select **Test with Web Services Explorer**.

You can expand the tree in the left pane until you see the list of functions that can be started in the service. If you choose the GetFlight function, the input parameters are listed in the right pane. In this case, you gave the flight number as input. After submitting the service request, you got all the relevant data from the database in the body of the output, as shown in Figure 11-38.

With the successful retrieval of the data, this shows the service works. Now, you can continue to build a web-based RUI to accept the input data and display the returned data.

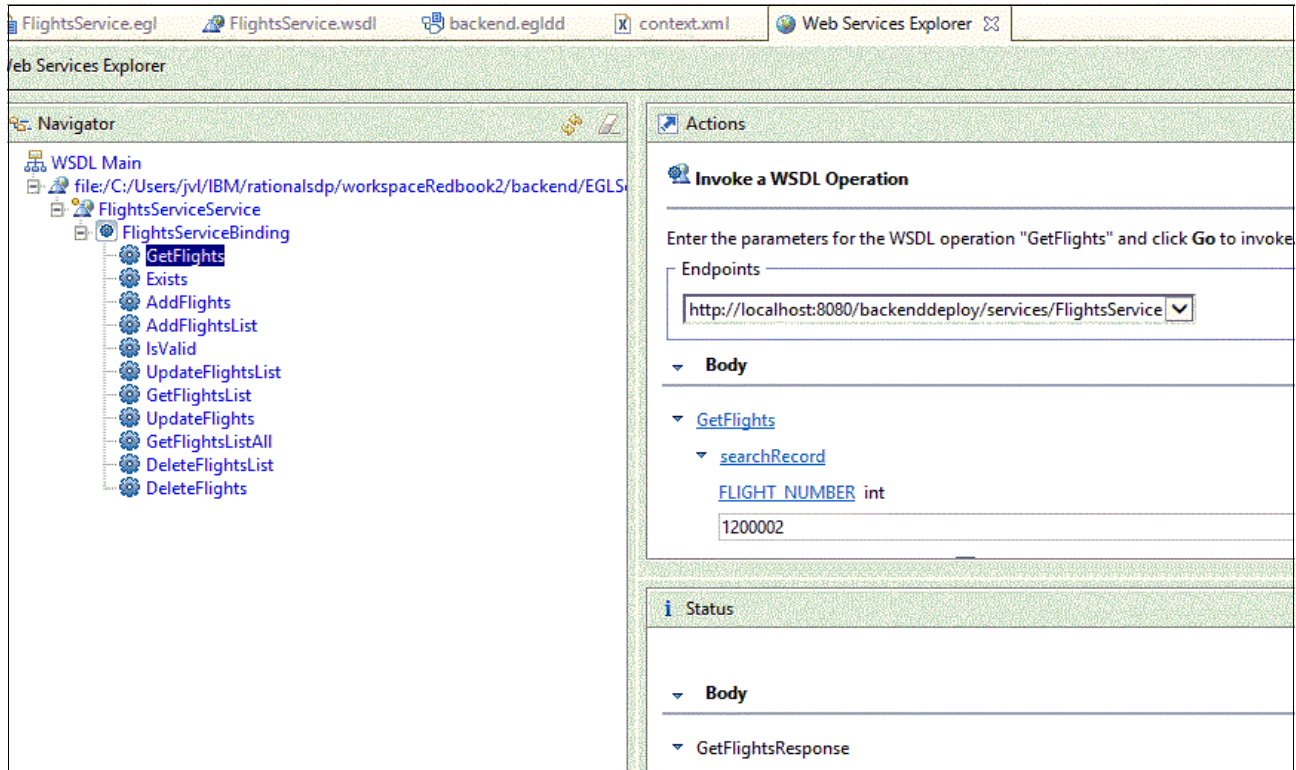


Figure 11-38 Testing the service with the web services explorer in IBM Rational Business Developer

11.5.4 Creating an EGL rich user interface front end

In this section, you create an EGL RUI project to contain the EGL source, which displays information from the services that you created in the data application access project. In the result, you can enter a flight number and request all the related information.

Click **File** → **New** → **EGL Project** or right-click in the Project Explorer view and select **New** → **Project ...** → **EGL** → **Project**. The EGL Project wizard window opens. Give your project a name and select **Rich UI Project**, as shown in Figure 11-39 on page 525. Click **Next**.

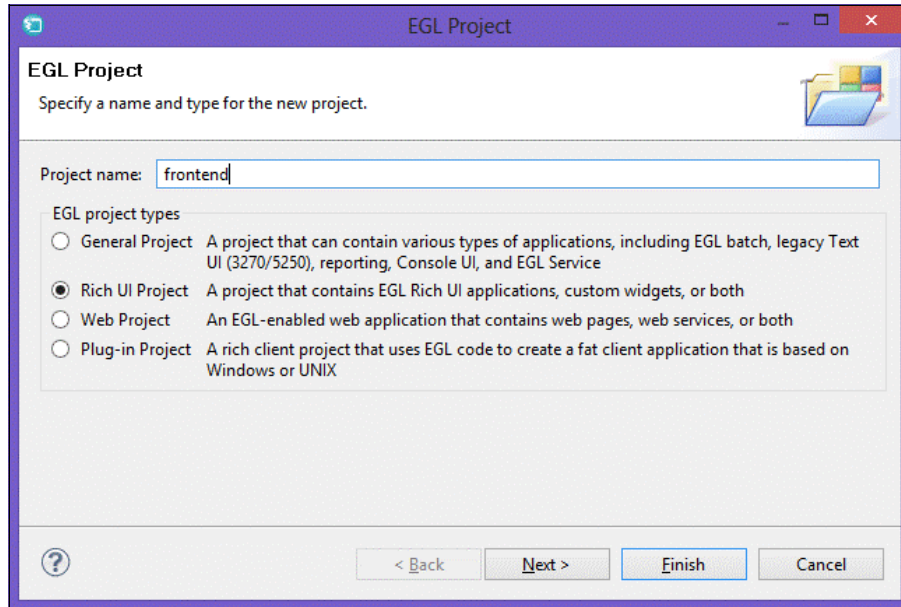


Figure 11-39 Naming your EGL project and selecting the project type

EGL RUI uses the widgets from the Dojo Toolkit. If you add an EGL RUI project to your workspace for the first time, these two projects are added automatically. These projects contain the widgets (for example, combination box or textbox) and related functions, which you can use on your window. For example, a calendar widget ensures that only a correct date is entered. This validation is done automatically for you by the widget, which means that you do not need to code special validation logic for certain basic components.

Click **Next** in the wizard to open the general settings window, as shown in Figure 11-40.

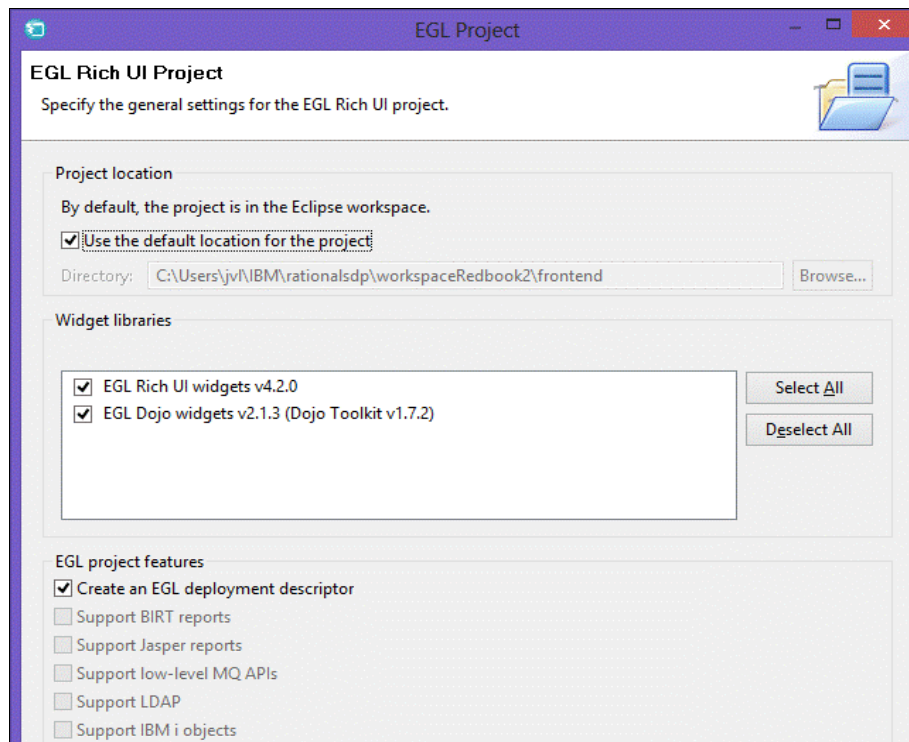


Figure 11-40 General settings of the EGL RUI project

In this project, you want to reuse the records, which were created in the back-end project. You need to create only an association between the RUI project and the back-end project. On the next window of the wizard, select the **backend** project. Now, you can reuse records from the back-end project in your front-end project, as shown in Figure 11-41. Click **Finish**.

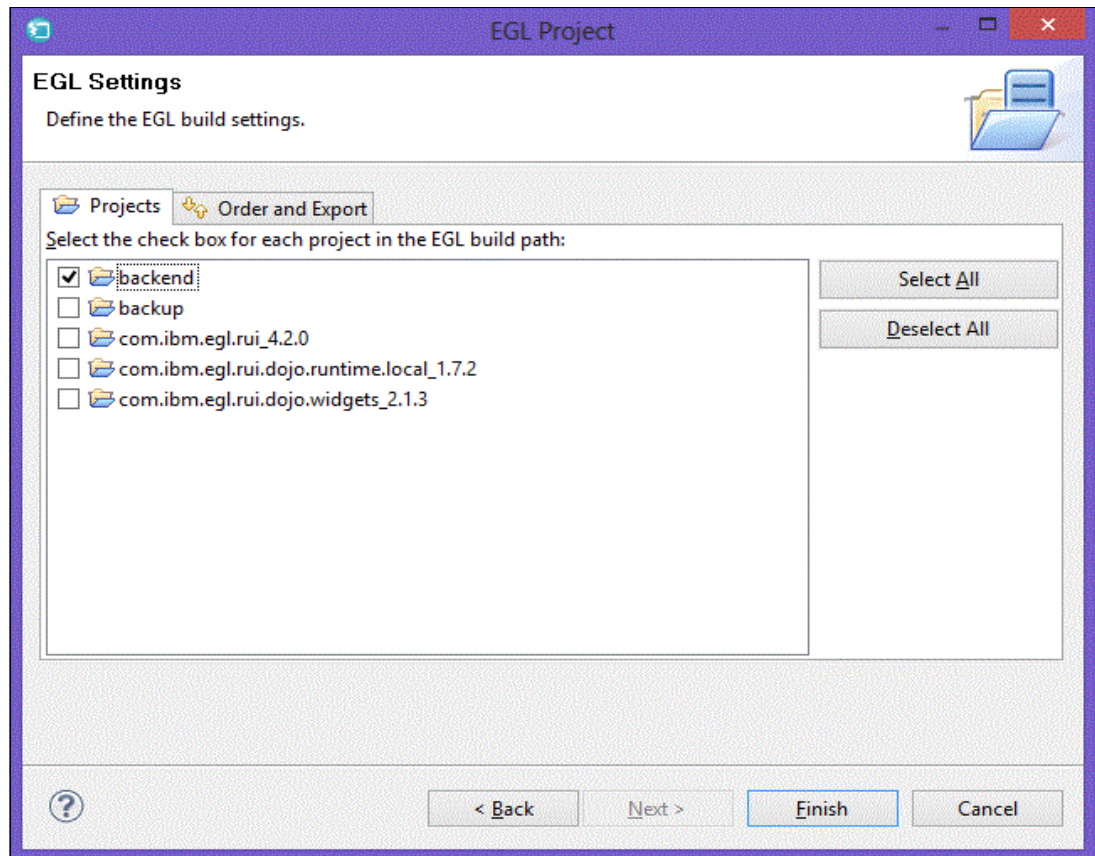


Figure 11-41 Selecting the back-end project

Right-click the newly created project and select **New** → **Other ...** → **EGL Rich UI Handler**. A rich UI handler (RUI Handler) is an EGL specification that represents both the widgets on a web page and the browser-side logic that runs when the user is interacting with the page.

When the window that is shown in Figure 11-42 on page 527 opens, choose the location where you want to keep this source. In this example, **flight400** was used as the package name to indicate that the package contains all the web pages that are associated with the “FLGHT400” tables.

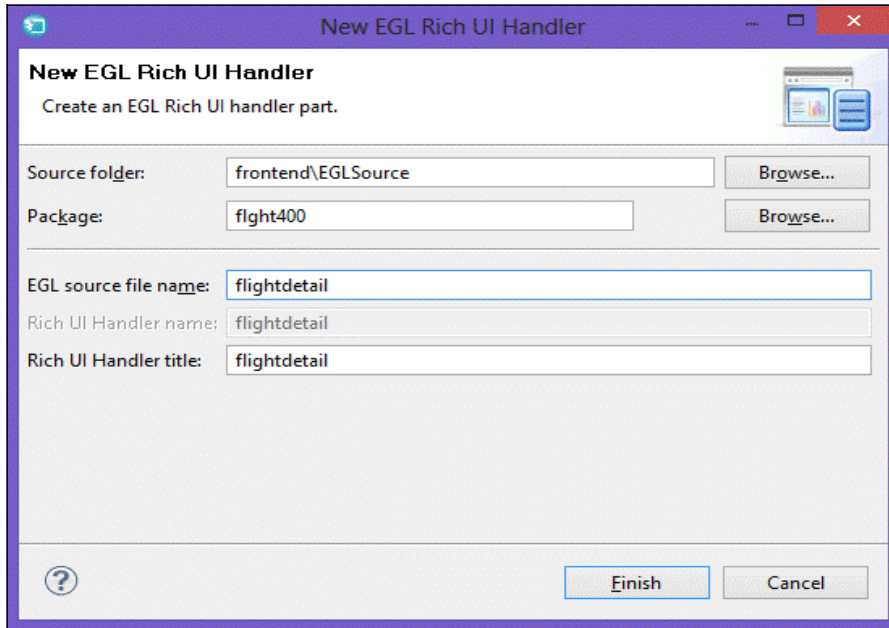


Figure 11-42 Location information for the EGL RUI Handler

When the RUI Handler is opened, it opens a special editor that is called the RUI Visual Editor, as shown in Figure 11-43. This editor allows you to visually see and build your web page and switch to working with the EGL source directly.

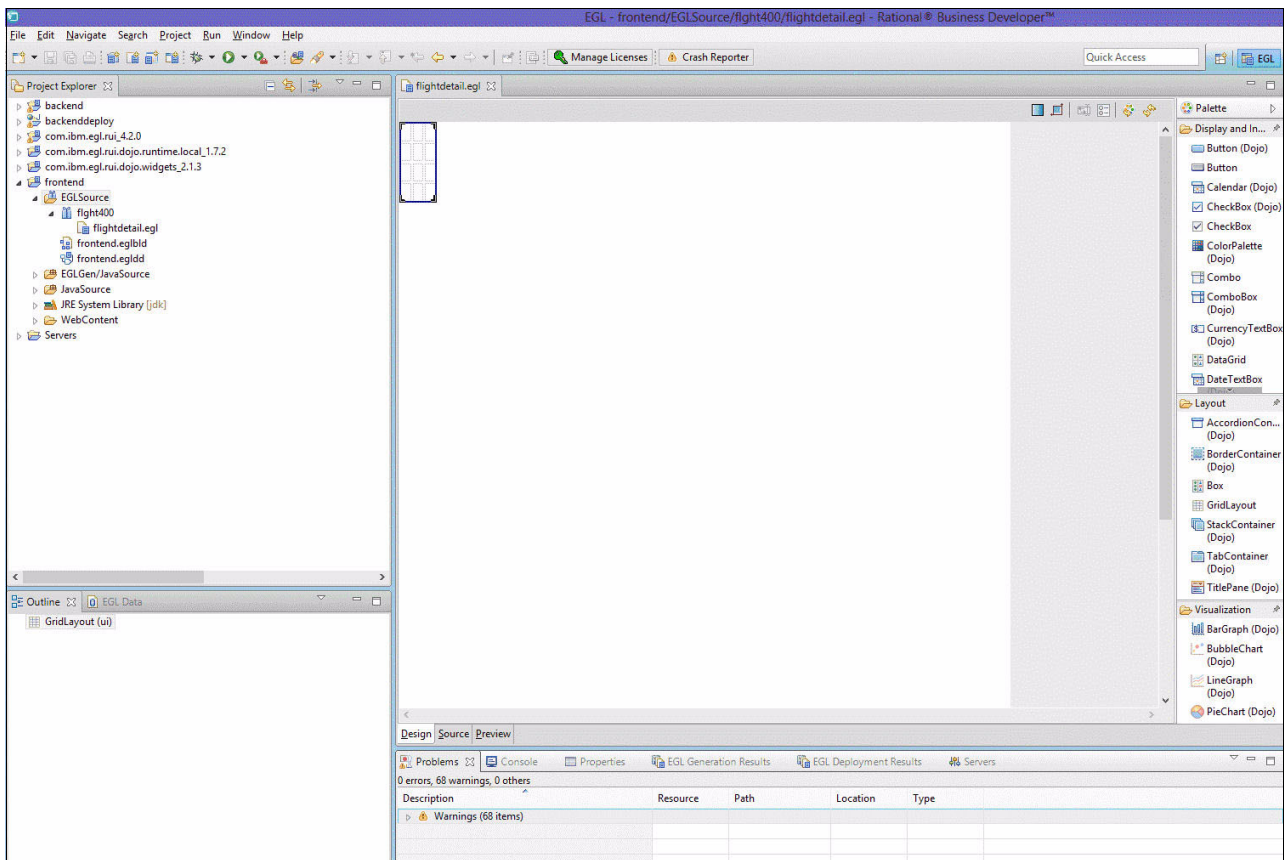


Figure 11-43 An empty web page is created in the RUI Visual Editor

At the bottom of the web page, you have three tabs:

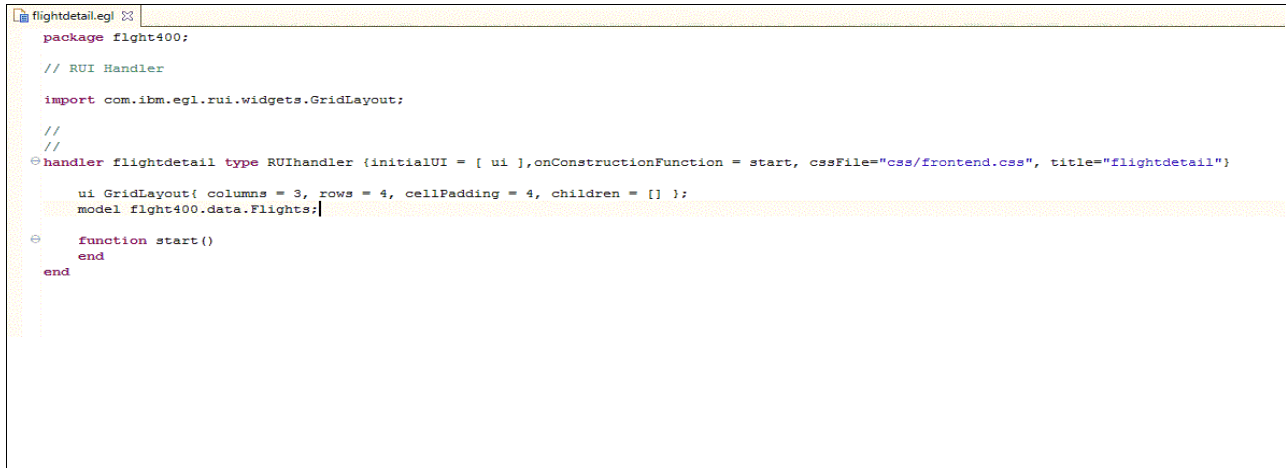
- ▶ Design tab: Use this tab to easily drag widgets from the palette onto your page.
- ▶ Source tab: Use this tab to change the code.
- ▶ Preview tab: Use this tab to see how the page looks at run time.

In Figure 11-45 on page 529, the design tab is shown. This web page has a grid layout, which is shown at the upper left corner. In the grid, you can add a widget so you can create a web page like it is shown at run time.

Because you already have an EGL record that represents the columns in our table, you can use this record to populate the page. To do this, on the Source tab, you can declare a basic record, which you need to interact with the service and populate the web page. This record was already created by using the data access application.

The statement in the following EGL source declares a variable that you can reference in the web page and pass to the service. The variable name is “model” and its type is the record that is created in the Data Access Application wizard. See Figure 11-44 for how the statement is inserted into the RUI handler source.

```
model flight400.data.Flights;
```



```
flightdetail.egl
package flight400;
// RUI Handler
import com.ibm.egl.rui.widgets.GridLayout;
//
//
handler flightdetail type RUIhandler (initialUI = [ ui ],onConstructionFunction = start, cssFile="css/frontend.css", title="flightdetail")
    ui GridLayout( columns = 3, rows = 4, cellPadding = 4, children = [ ] );
    model flight400.data.Flights;
function start()
end
end
```

Figure 11-44 The source view with the record declaration

If you go to the Design view, you can drag the record from the EGL data view in to your new web page, as shown in Figure 11-45.

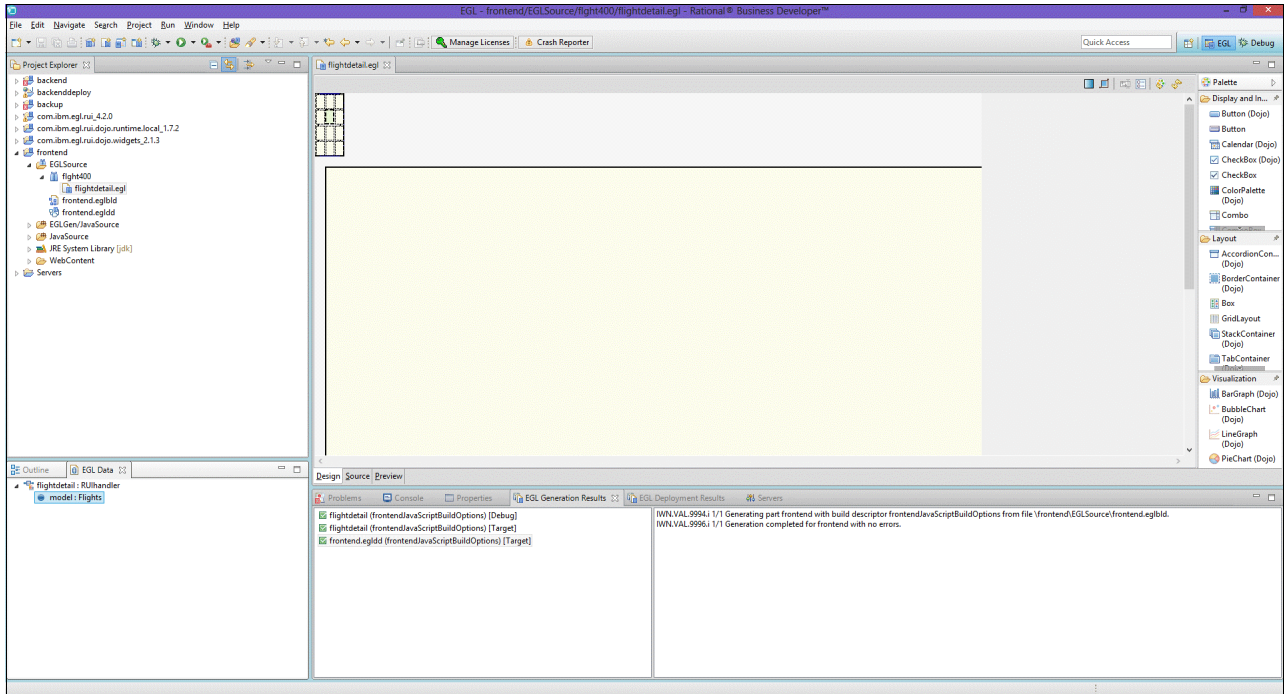


Figure 11-45 Dragging a record from the EGL data view to the design view

A wizard opens. If you select **Editable data**, as shown in Figure 11-46, you can enter a flight number from which you want to see the data.

In the column Widget Type, every field in this example is set to DojoTextField, as suggested by EGL based on the field's data type. However, the widget type can be changed. For example, a field that is used to enter a password can be set to PasswordTextField. If there is a list of available values, you can choose a drop-down box or other types as needed. Figure 11-46 shows an example.

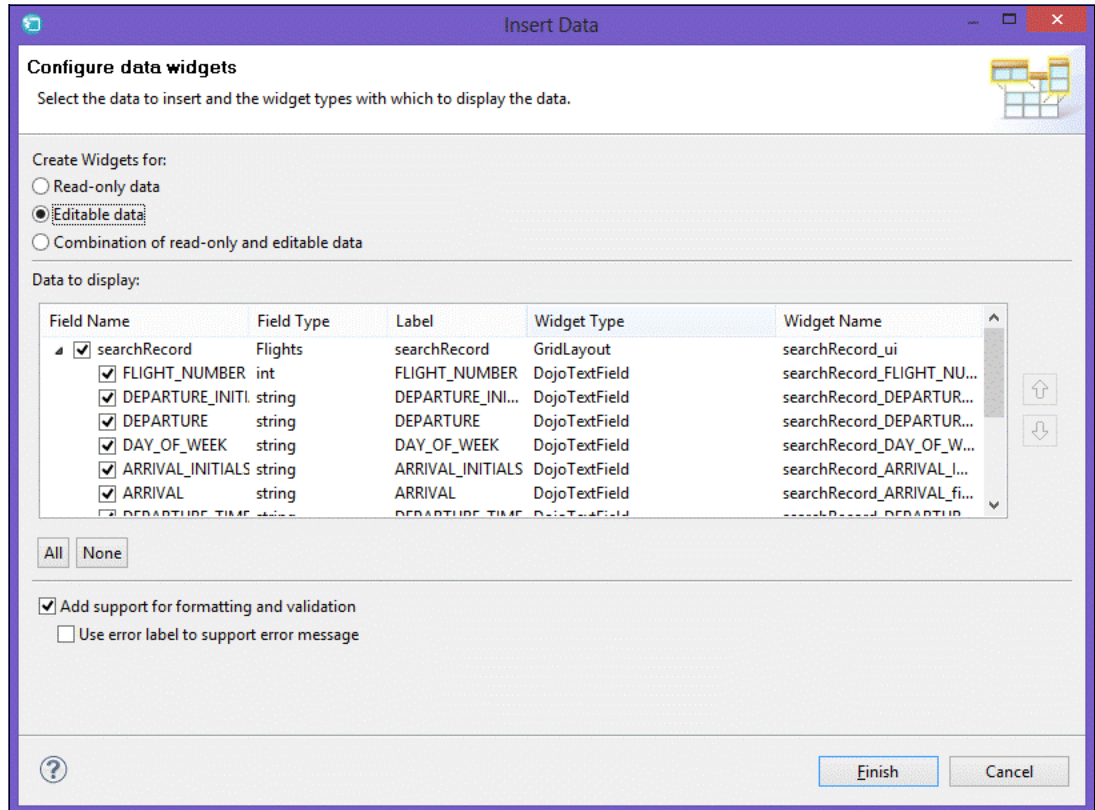


Figure 11-46 Configuring the data widgets

Assuming the default values are accepted, for every field of the record, a label and a field is created, as shown in Figure 11-47 on page 531.

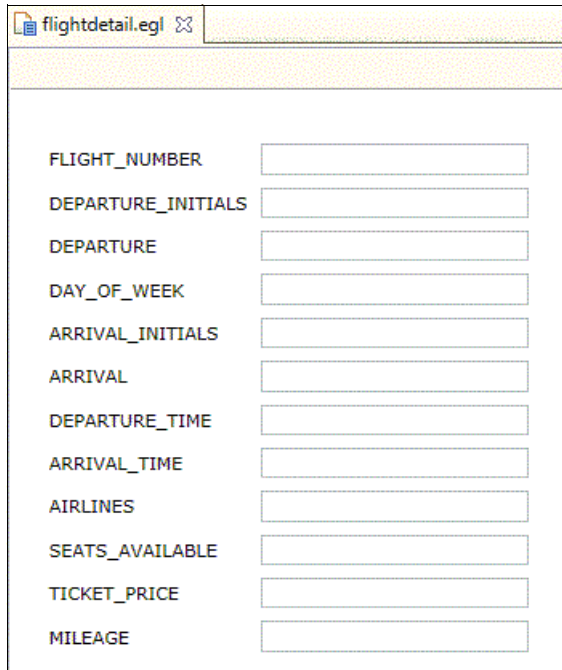


Figure 11-47 The result of dragging a record

Figure 11-48 shows the source tab for your page. You can see that much code is generated. All the fields are in the source of the project. Additionally, code for a Model View Controller (MVC) type handler is generated. You use this code later.

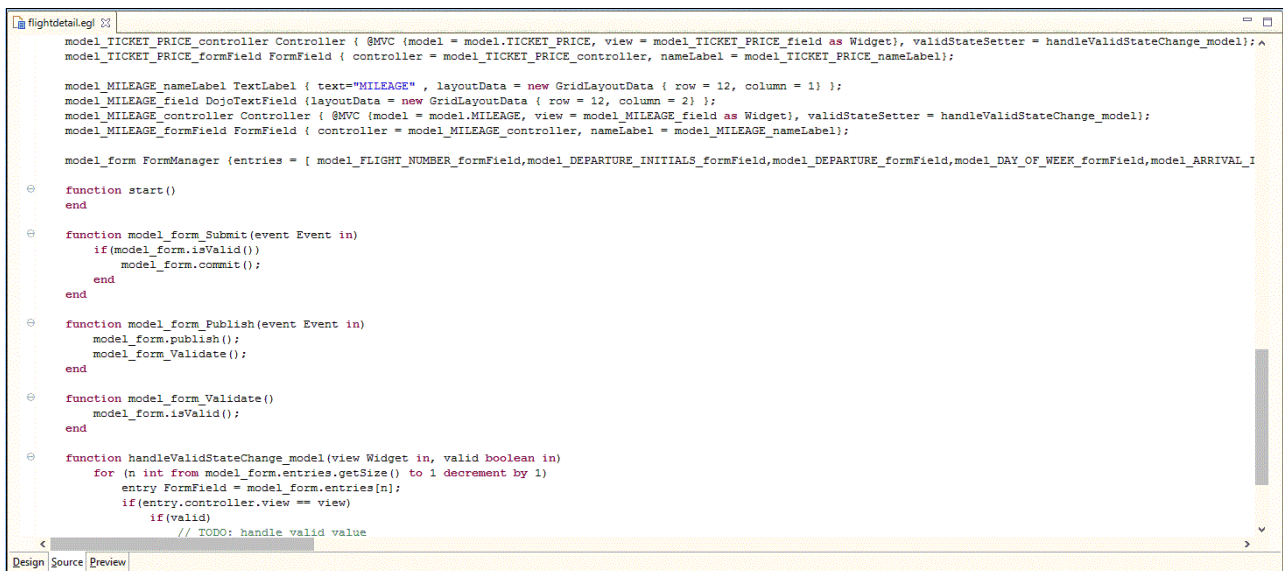


Figure 11-48 The generated code of your web page

If you copy the WSDL file that you created previously to the EGL RUI project, you can use it in your front-end project to create an interface to your web service. Other WSDL files can also be used if more than one service needs to be started.

EGL provides a wizard to create the interface and also update what is called a *service binding* in the EGL deployment descriptor file. The EGL deployment descriptor is the file with extension .egldd in your project. It contains specifications about how the project must be deployed and information about how services should be started. By adding the reference to the WSDL, you are enabling the RUI project to use and start the service with SOAP.

Figure 11-49 and Figure 11-50 on page 533 show the wizard, including the information to specify the service binding (Figure 11-49) and to create the EGL interface to the web service (Figure 11-50 on page 533).

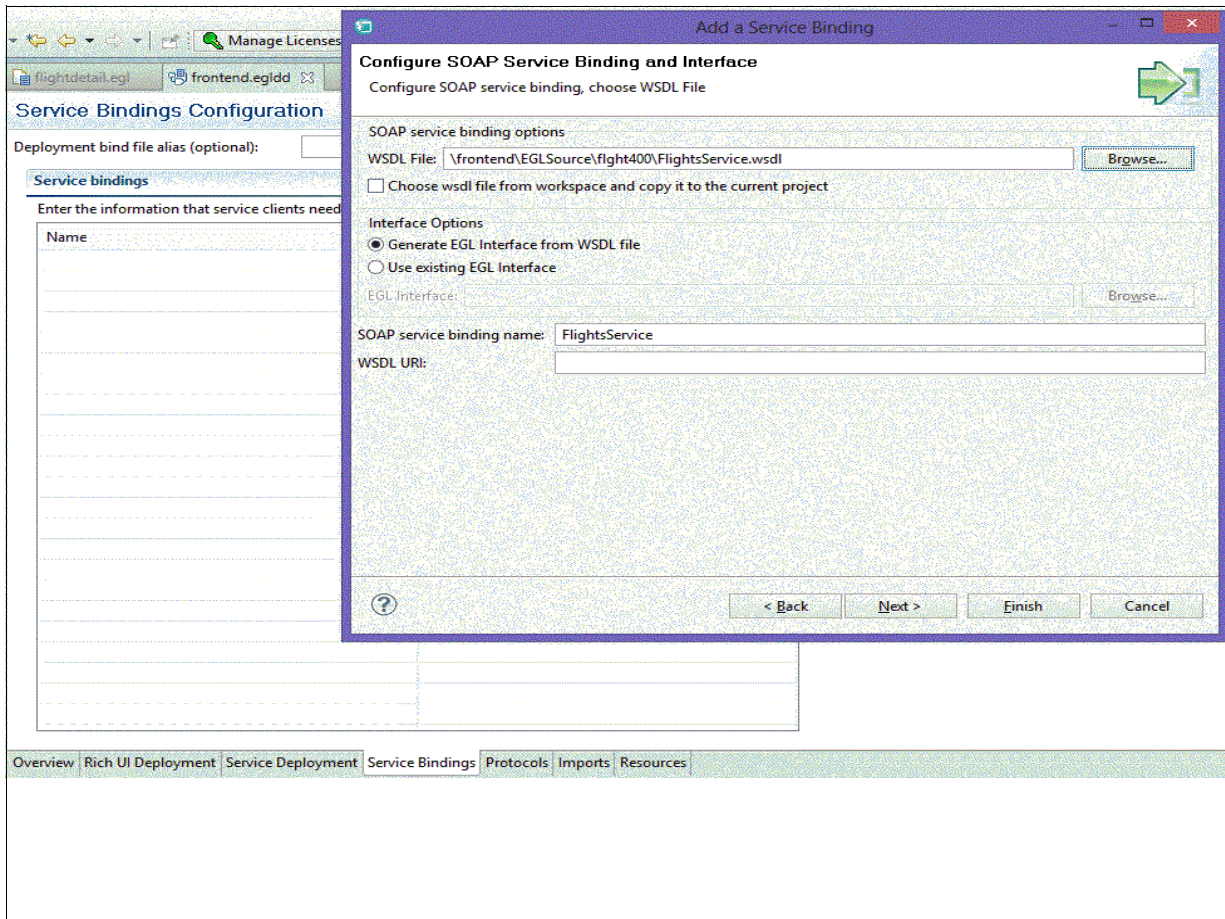


Figure 11-49 Adding a service binding in your deployment descriptor

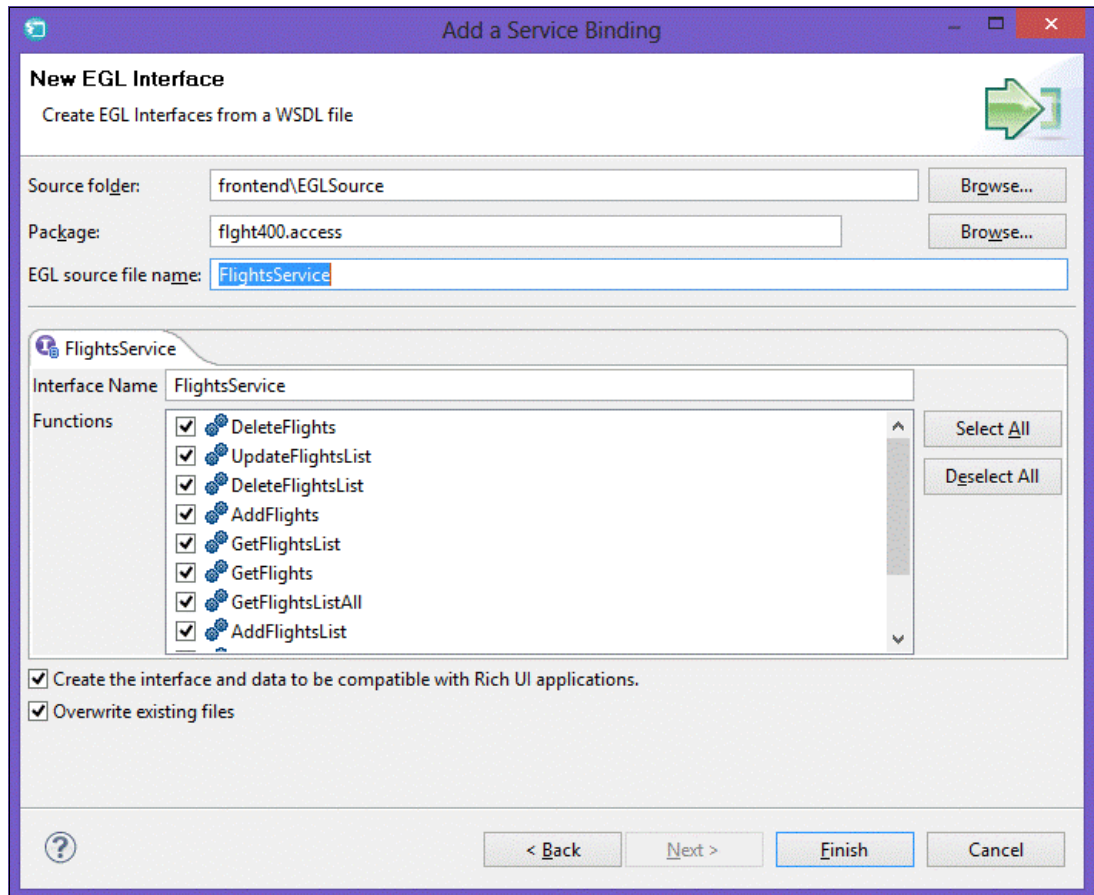


Figure 11-50 The specifications of the EGL interface

In your web page, you can add the following code:

```
FlightService FlightService {@bindService};
```

With this code, you declare a service variable. Through this variable, you can now start any function within it from the web page.

To start one of the functions in the service, you now can add an event to be started when some action is taken against a widget in the page. To do this, in the Design view, click the Flight Number field. In the Properties view, you can add an action to it. For example, if you are going to choose the action `onFocusLost` (which means that if you enter a value and then tab out of the field that the function is run). Figure 11-51 shows an example.

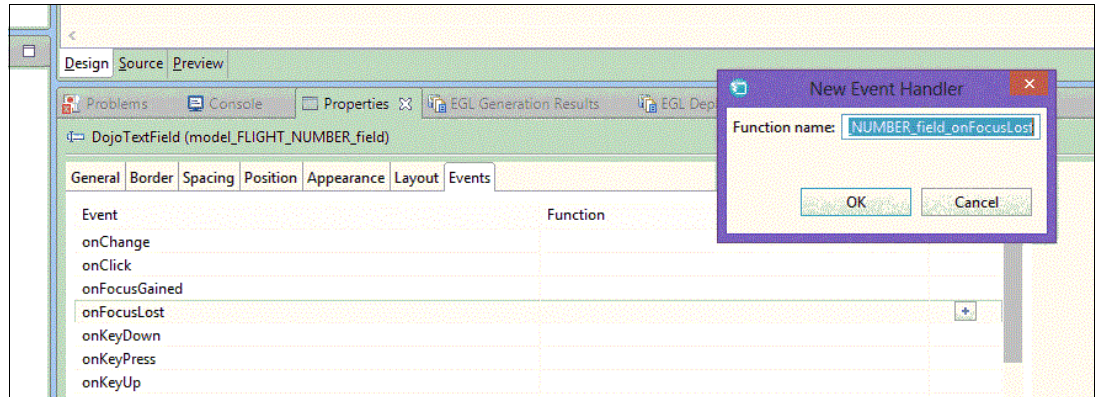


Figure 11-51 Creating the Event Handler in the Properties tab

In the source of the RUI page, the following code is created, which is started when the user tabs out of the field:

```
Function model_FLIGHT_NUMBER_field_onFocusLost(event Event in)
End
```

You can add some additional code to this function:

```
Function model_FLIGHT_NUMBER_field_onFocusLost(event Event in)
    model_form.commit();
    call FlightsService.GetFlights(model, status) returning to myCallback
    onException myExceptionHandler;
end
```

The `commit` line moves the information from the window into the appropriate record, which is the `model` variable in this case.

The `call` line calls the service for this record. If you right-click the `call` line, Rational Business Developer automatically creates the callback and exception functions.

A callback function is the function that is started after the service runs. It gets the returning parameters as input and handles them. The `onException` function is called when something goes wrong, such as the service not being found.

In the automatically created callback function, you can also add these lines:

```
Function myCallback(searchRecord Flights in, status StatusRec in)
    this.model = searchRecord;
    this.model_form.publish();
end
```

The first line moves the returned record to the variable for the web page. The second line updates the information on the window.

This code uses a Model-View-Controller (MVC) paradigm to separate the user interaction and the representation of information. In this example, you can focus on the record, enter the correct data, and the MVC helps you display it.

If you run the web page, you can enter a flight number, as shown in Figure 11-52. You can do this as a deployed application, or you also can test this by using the preview mode. Using the preview mode means that you can easily test and change your code without waiting to completely deploy the application to an application server.

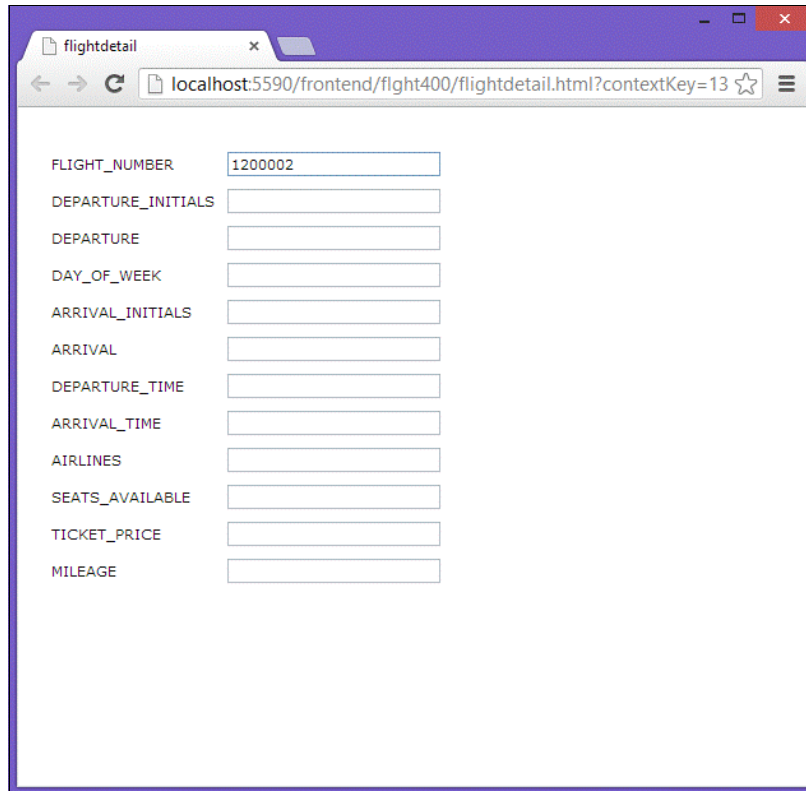


Figure 11-52 The web page at run time

If you move out of the field, the requested information is displayed (Figure 11-53).

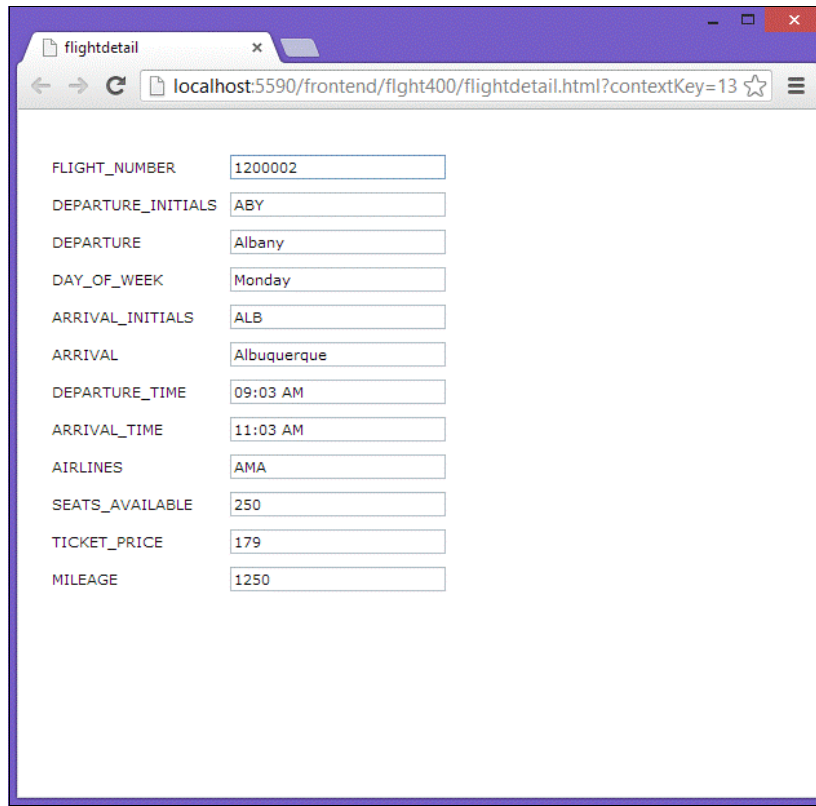


Figure 11-53 The result of your web page

11.6 EGL: Calling external programs

From EGL programs and services running as Java, you can call external programs such as RPG, COBOL, CL, or other programs that are running on an IBM i. A call from EGL to RPG is often done to use the existing business logic in the RPG program.

For calls, EGL uses the Java toolbox (jt400.jar) and the remote program call function.

The programmer does not need to understand all the intricacies of how to use the JT400 toolkit. Instead, they have to code only a simple call statement, as shown below, and provide a few specifications about how the RPG program should be called.

A simple statement looks like this:

```
call "aProgram" (parameter1, parameter2);
```

11.6.1 Using linkage to call an RPG program

To specify information such as the system that the RPG program is running on and what library it is in, you must specify linkage information for EGL.

To get started, you must add the jt400.jar file to the project's class path. This is done by right-clicking the project, selecting **Properties**, clicking **Java Build Path**, clicking the **Libraries** tab, and clicking the button to add an external JAR file. Browse to the jt400.jar, as shown in Figure 11-54 on page 537.

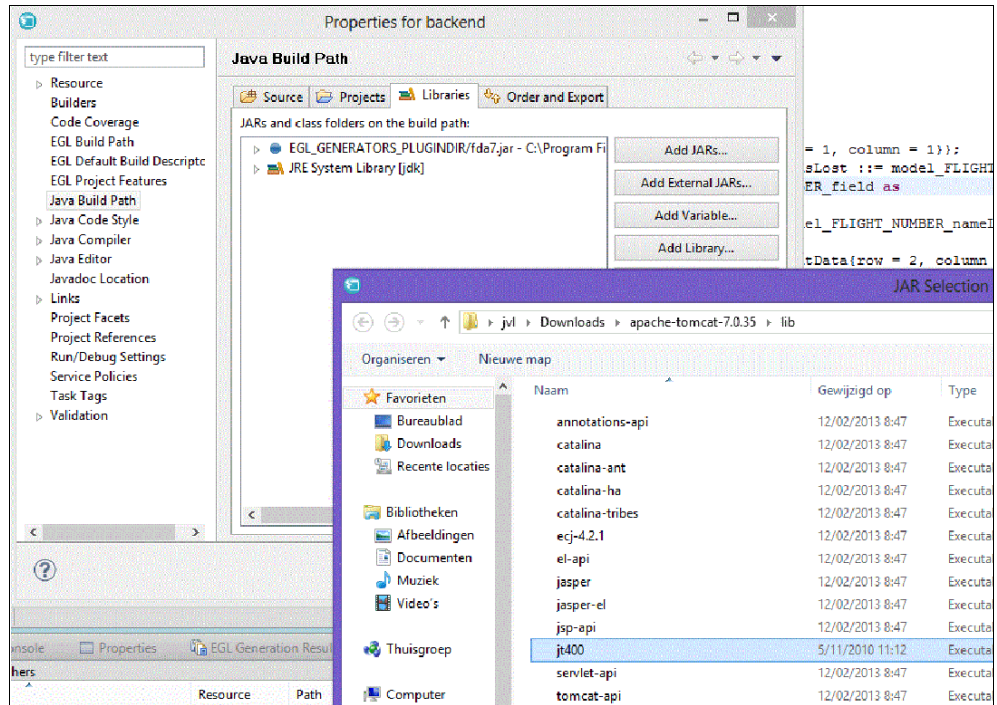


Figure 11-54 Adding jt400.jar to your project

Instead of putting the file in the Java build path, you can also copy the file to the `WebContent\WEB-INF\lib` folder. This approach is needed if the program or service is running on an application server.

In your workbench, there is a build descriptor file. The content of this file controls the generation process. If you followed the previous steps, under EGLSource in the backend project there is a file that is called backend.eglbuild. This file is the default build descriptor for this project. You can double-click to open it. The file contains some values, as shown in Figure 11-55.

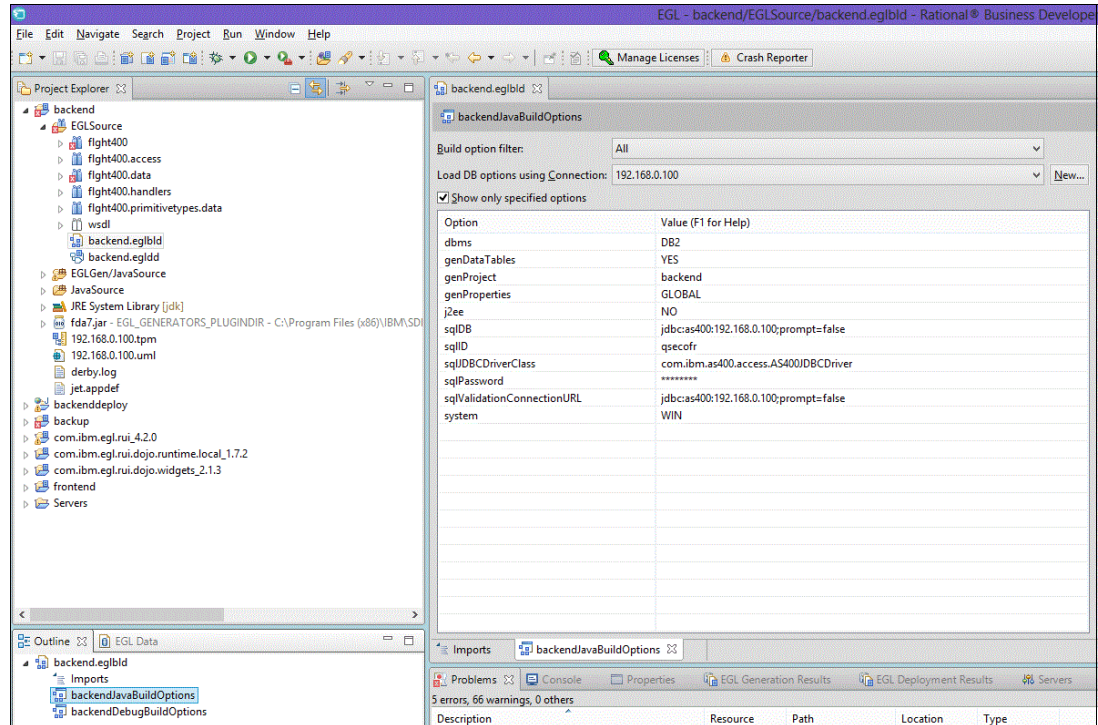


Figure 11-55 An example of a build descriptor

In the build descriptor file, you must add a calllink element for every program that you want to call. A calllink is contained in a linkage part. To create a linkage part, as shown in Figure 11-56 on page 539, in the outline view, right-click the build descriptor and select **Add Part...**

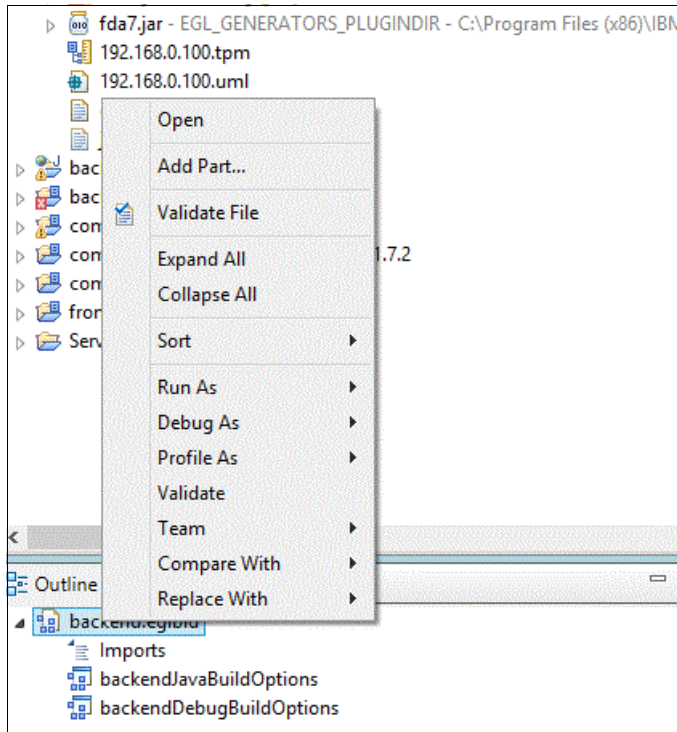


Figure 11-56 Menu in the outline view of the build descriptor

A wizard opens. In the first window, select **Linkage Options**, as shown in Figure 11-57.

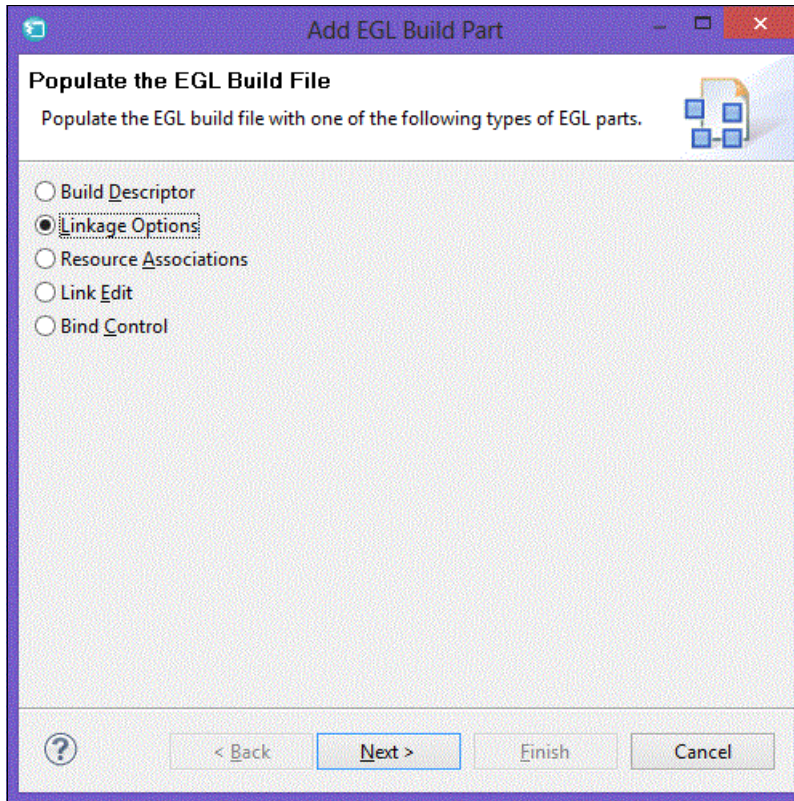


Figure 11-57 Adding linkage options to your build descriptor

As shown in Figure 11-58, on the last window of this wizard, give the part a name (and a description if you want) and click **Finish**.

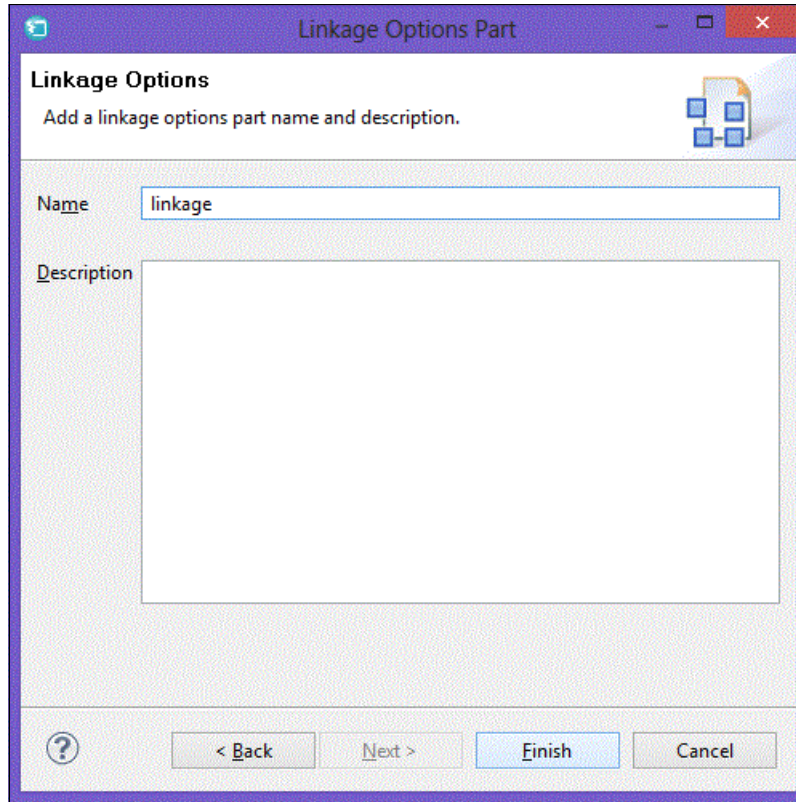


Figure 11-58 Giving a name to the linkage options

A new linkage part without elements is created in the build descriptor, as shown in Figure 11-59.

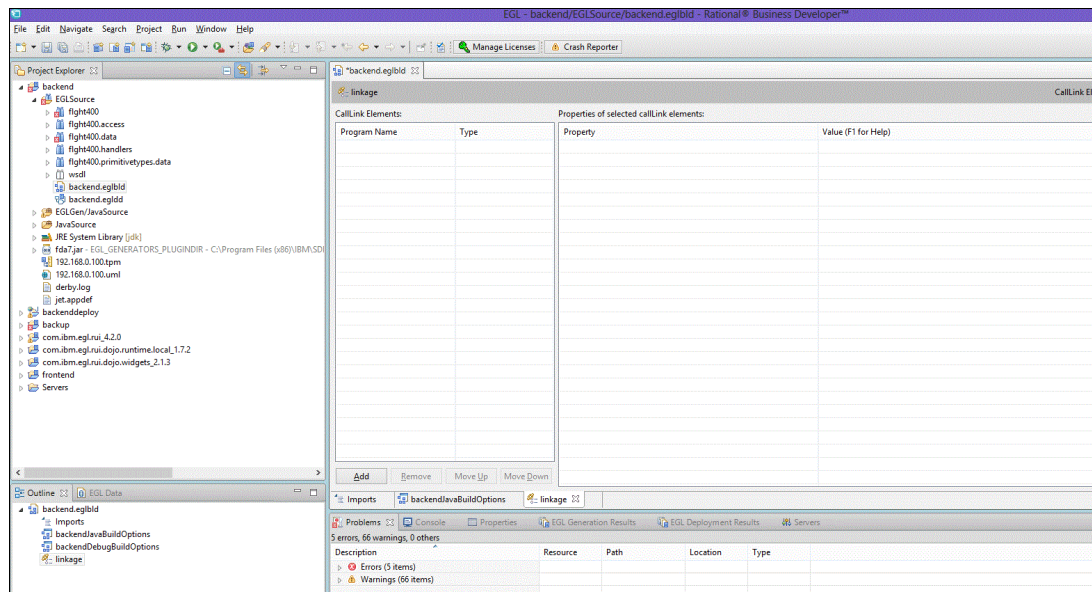


Figure 11-59 A new empty linkage part

Click **Add** and give a name to the program in the left column. Set the Type to remoteCall and change the properties, as shown in Figure 11-60. These properties provide information such as the location of the IBM i system, the library that contains the program to be called, what conversion tables to use when converting the data from Unicode to EBCDIC, and other values.

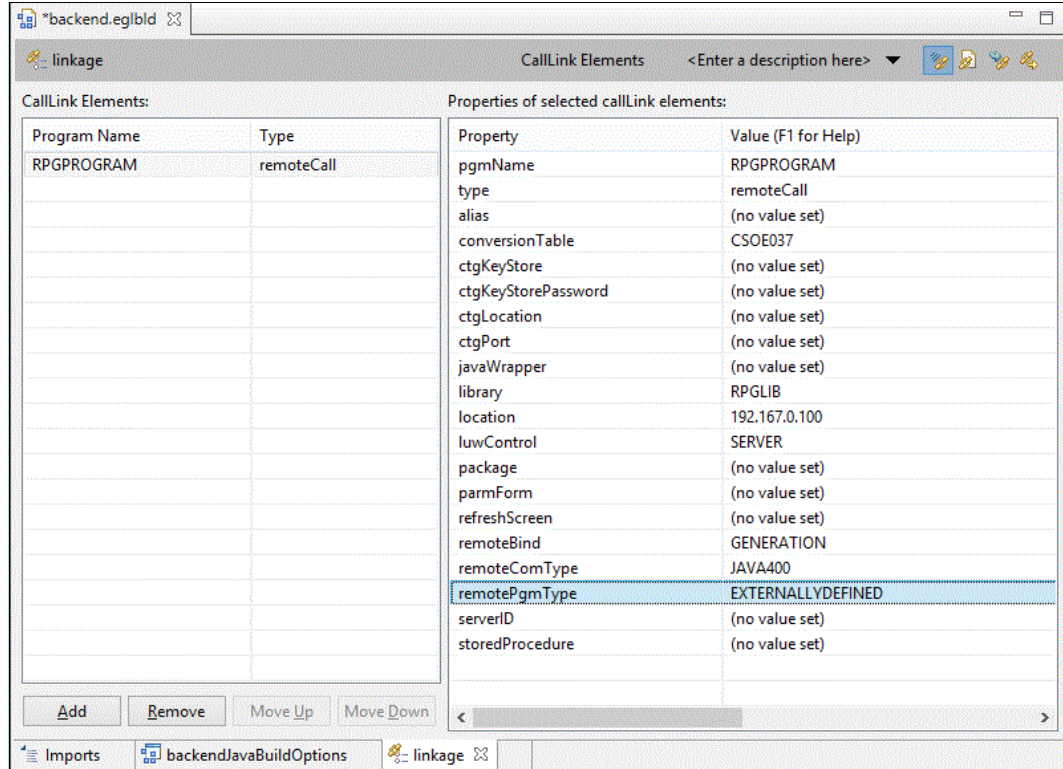


Figure 11-60 An example of a call to an RPG program

In the outline view, double-click the build descriptor to generate the Java for your services, as shown in Figure 11-61. The build descriptor opens again. Now, you can specify your newly created linkage part in the linkage build descriptor (see Figure 11-62). You might have to clear **Show only specified options** for this option to display.

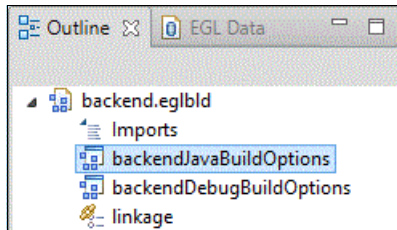


Figure 11-61 Outline view

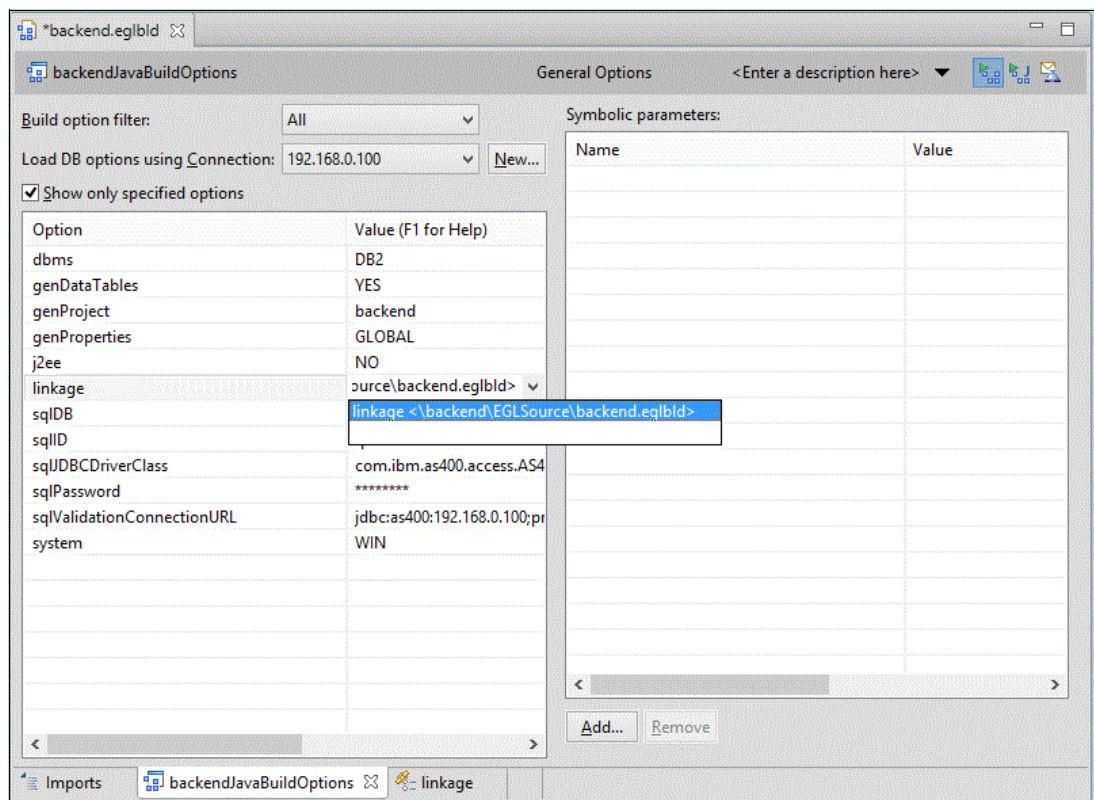


Figure 11-62 An example of a call to an RPG program

In your code, you can now enter the following line:

```
call "RPGPROGRAM" (aParameter);
```

This line causes the RPG program to be called and run on the specified IBM i system. The variable that is named aParameter represents the data that is expected as the input and output parameters for the RPG program. These can be a single field or a structure.

The data that is returned can then be passed back to an EGL RUI handler through a return from the service and can be used to populate the page.

This section provided an overview of how a call can be done to the IBM i. EGL helps provide much more information about this subject. There are also many tutorials and samples in the EGL Cafe, found at the following website:

<http://www.ibm.com/developerworks/rational/products/rbde/>

11.7 Benefits of EGL for IBM i

EGL has many capabilities that enable an IT organization to modernize its applications and write new applications in a single approach. Here are the overall benefits of EGL:

- ▶ End-to-end or web-to-database development in a single tool and single language
- ▶ Broad capabilities to fit any application architecture
- ▶ Easy integration with IBM i for reuse or access of existing RPG programs and data
- ▶ Usage of modern technologies for development and modernization of existing applications
- ▶ High productivity through wizards, visual editors, and other tools
- ▶ A short learning curve for both traditional and new programmers, regardless of the application architecture and requirements



PHP

This chapter describes the open source web scripting language that is called PHP and approaches the description from the perspective of an experienced IBM i developer. Because there are many books about how to use PHP, the fundamentals of PHP are not heavily described in this chapter. Instead, the chapter focuses on many of the aspects of the PHP experience as it applies to IBM i.

12.1 What is PHP

PHP Hypertext Preprocessor (PHP) is an open source, server-side scripting language that controls more than 244 million websites¹ around the world and also runs natively on IBM i. PHP's popularity is due in part to its easy-to-use syntax and an incredibly scalable and powerful code base. With a little code, a lot can be accomplished, which is the hallmark of a good scripting language.

12.1.1 Open source

The term *open source* can mean many things to different people. As a language, no one truly owns PHP. It is a language that is developed by the PHP community for the PHP community. The evolution of the language is a democratic process with thousands of volunteers around the world contributing code, documentation, and project management in support of the world's most popular web language. Anyone with a C compiler can download the source code for PHP and compile their own copy to run their websites. The inexpensive nature of PHP initially made it an ideal choice for startup companies and not-for-profit entities worldwide. But in the last several years, commercial support entities like Zend Technologies have brought PHP into the enterprise by offering an optional support model. This gives organizations the tremendous cost savings of PHP from an application development standpoint, and gives customers peace of mind in knowing that, if there is a problem, they are not alone and can reach out for help.

The nature of PHP's open source and fairly democratic roots represents a fundamental difference when comparing PHP to RPG. IBM protects compatibility with earlier versions in a zealous manner and this has led to a long and prosperous history. The PHP community makes no such guarantee. It is possible that a function that is used in a PHP script today might not be available tomorrow. Remember, PHP is maintained by the community for the community. It is not in the general interest of the community to remove features without a good reason. Often, when something is removed, it is because of security.

12.1.2 Documentation

There are many books about how to learn PHP or build applications using PHP, so this chapter avoids most of the details of the language in favor of an accelerated focus on how PHP can be used by IBM i developers of all levels. You do not need to buy books or documentation for every PHP function. There is a website that is managed by the core PHP development team where nearly every PHP function is defined. See the following website:

<http://php.net>

Like the PHP language, the <http://php.net> site is managed by the community for the community, and there are several aspects to the site. The two most useful sections for beginners are “Language Reference” and “Function Reference”. The “Language Reference” section describes the many aspects of PHP, such as basic syntax and variable usage, and the “Function Reference” section provides detailed definitions and examples of the thousands of functions that make up the language.

In Figure 12-1 on page 547, which shows the DB2_Connect function page of <http://php.net>, several features are illustrated. The skeleton diagram of the function call is in the main section of the page in addition to the list of similar DB2 commands that are listed on the left side of the page. The search box on the web page allows for quick and easy prompting and retrieval of any of the more than 4,000 PHP functions.

¹ Source: <http://news.netcraft.com/archives/2013/01/07/january-2013-web-server-survey-2.html>

Figure 12-1 PHP.net PHP function search

12.1.3 What does PHP look like

This section does not provide details about the fundamentals of PHP because there are many books about the subject. However, this section introduces the language and its basic syntax. This section also explores many examples of PHP usage on IBM i. Figure 12-2 shows a simple example of a PHP script that displays today's date in two formats. As you can see from the code sample, PHP can be a brief language. This is true for many scripting languages, where the objective is to get tasks done. However, PHP can also be elegant and verbose when needed. Enterprise implementations of PHP use code libraries and frameworks, both of which can provide productivity benefits to the typical developer.

```

1 <?php
2
3     $today = date ( 'r' );
4     echo "Today is $today<br /><br />";
5
6     $today = date ( 'm/d/y' );
7     echo "Today is $today";
8
9 ?>

```

Figure 12-2 Two dates PHP example

Figure 12-3 shows the output for Figure 12-2.

```

Today is Tue, 11 Feb 2014 06:10:16 -0800

Today is 02/11/14

```

Figure 12-3 Browser output for two dates example

12.1.4 How PHP works

Figure 12-2 on page 547 illustrates the simple approach that PHP takes to web design. The whole purpose of PHP as a server-centric web language is to produce something the browser can digest (Figure 12-3 on page 547). This might seem like an oversimplification, but this concept is refined later in this chapter. PHP, like RPG for example, runs 100% on the server. The only content that comes from the server is HTML, JavaScript, and so on. These are technologies that only the browser understands.

In this brief example, the date is displayed in two formats with a small amount of coding that is required for the output. Code brevity is one of the hallmarks of the PHP programming language. Like RPG, a small amount of code can go a long way.

12.1.5 Why use PHP

This section explores some of the many reasons why PHP is attractive to the greater IBM i community. But the most compelling reason to use PHP on IBM i is apparent when you start to learn the language. PHP supports essentially three models of development: inline coding, procedural, and object-oriented. Each of these three models is described in this section.

Learning curve

Many IBM i developers started out as inline developers (RPG II) and moved up to procedural development (RPG III, ILE, or COBOL). As a result, some programmers struggled to embrace more “modern” languages because some of the newer technologies assume a knowledge of object-oriented (OO) structures. Without a solid understanding of OO, many concepts become difficult to grasp and suddenly make language adoption difficult. PHP helps developers by giving them the option to start with the techniques with which they are most comfortable. The procedural model of PHP development looks much like RPG with subprocedures or even subroutines. Many RPG developers have found PHP to be an approachable language. Even more encouraging is that the community around these developers is forgiving. Overall, the PHP community seems to be forgiving and understanding for new developers.

Equally important as an adoption point on the learning curve is the point where the developer becomes productive. PHP developers start being productive almost immediately, thanks to the forgiving nature of the code and flexibility of the language. This is not to say that the first PHP scripts are works of art. Think of your first RPG or COBOL program. It worked and it might not have been pretty, but it probably accomplished a task. Developers learning and growing with PHP can do the same.

Perfect fit

Because RPG and IBM i lack a native graphical user interface (GUI), PHP for application development addresses this need. RPG programmers who typically are more focused on business processes than the minutia of coding can find the addition of PHP a welcome asset in the application modernization process. Although PHP can handle the web user interface of the front end of an application, it can make calls into heritage RPG and COBOL applications, thus ensuring companies around the world can continue to use years of investment in application development. As a programmer grows with PHP, so does their skill set, and access method.

Community

There are millions of PHP developers around the world and one of the hallmarks of this vast community is a love of sharing code examples. Look around the internet and you can see countless examples of PHP doing many things, from data validation to JavaScript integration. In many cases, these samples can be used and reused at no charge. In addition to being useful snippets of code, copying these samples can help a developer learn the basic and more advanced features of PHP. This is the same way many programmers have learned RPG and COBOL, that is, by working with the code and not just reading a book.

One of the more popular terms is Other Peoples Objects (OPO) or Other People's Code (OPC). Developers often learn PHP by using OPO or OPC. This means many developers start using objects as a "black box". A curious developer, once comfortable with the black box, begins peering inside to see how it works. The first time a developer examines the inside of a class, they might be understanding basic interfaces. The next time, they might trace the flow of logic. Soon, the developer is making changes and creating their own objects. There are many websites that are dedicated to the art of sharing pieces of code or entire applications. Here are a few examples:

- ▶ PHP Classes:
<http://www.PHPclasses.org>
- ▶ HotScripts:
<http://www.hotscripts.com>
- ▶ PHP Freaks:
<http://www.phpfreaks.com>
- ▶ SourceForge:
<http://www.sourceforge.net>

Scalability

One of the luxuries IBM i developers have taken for granted is the sheer scalability of applications running natively that support their business applications. Adding another user to a system is often a simple task and few people worry about how many users they add. Over time, you might discover that system resources, although in many ways are self-managing, are occasionally over-taxed because of organic business growth or acquisition. IBM simplifies the management of this situation by abstracting the operating system from the user environment, which is not reviewed here. However, what is reviewed is an overly simple approach to scaling up, which is to simply order more hardware.

Because PHP was born in the Linux world, much discussion about scaling happens "horizontally". This means that many shops improve their throughput by adding more servers. For IBM i, IBM has simplified this process tremendously. Scaling on IBM i tends to be referred to as "vertical" scaling because one server tends to get more resources, such as processors or memory. If there is a need to go to a new model class, a simple save and restore brings a customer up to where they need to be to process transactions efficiently.

As a result of the horizontal scaling aspect of PHP on Intel style servers, the environment tends to have something that is known as *clustering*. Clusters allow other servers to step in when another server fails. Because IBM i tends to be highly reliable and can scale vertically, there is no need for clustering and, therefore, it is not available on IBM i as of this writing.

12.1.6 Who is Zend

PHP began life as the hobbyist creation of Rasmus Lerdorf, a Danish developer who needed to manage his HTML web pages. As a C engineer, Rasmus had experience with PERL. Although using either of these languages as a CGI engine was possible, he was not impressed with the lack of convenience. So, he started working on the first version of PHP, which he called Personal Home Page/Forms Interpreter. Rasmus released this hobby project as an open source project and managed the first two versions of the language.

A few years later, two students at Technion University in Tel Aviv were approached by their college advisor to get going with a project or they might not graduate. As they looked at each other, their advisor offered that there was this open source project that was called PHP and that they might look at improving the language. They immediately set out to download the project and looked it over. Then, they sent a note to Rasmus indicating that they wanted to revise the parser (real-time interpreter of PHP) to improve speed and function. Rasmus told them to go forward and have a great time.

Commercial support

Those two university students were Andi Gutmans and Zeev Suraski, the ZE and ND of Zend Technologies. Andi and Zeev have been at the helm of PHP as an open source project through versions three and four. Today, at Version 5, they are still heavily involved, but others have stepped forward to provide the primary leadership.

Andi and Zeev created Zend as a commercial entity to provide support to enterprise organizations using PHP as a language to build business applications and informational websites.

IBM Business Partner

In 2005, IBM was petitioned by several of their advisory groups to explore an easy-to-use web scripting language. IBM Net.Data was wildly popular but stabilized several years earlier. IBM approached Zend to bring PHP to IBM i and the resulting partnership introduced Zend Core, and later Zend Server for IBM i, which is distributed as part of the IBM i operating system. The current product as of this writing is Zend Server 6 and it is used for most of the remaining exhibits.

12.1.7 PHP environment setup on IBM i

The ideal environment might not always be the most cost-effective, but it should be considered seriously when you are implementing new technologies, regardless of the language. For PHP, a single LPAR should support a single implementation of Zend Server. Zend Server can support multiple applications and can be customized to provide support for each application differently, depending on the version of Zend Server that is selected.

How many environments

A preferred practice for business environments is the “three environment approach”, with each on a separate Logical Partition (LPAR) of IBM i. Some companies have as few as one or as many as five or six. There is no hard and fast rule for how many environments to have. Here are three environments:

- ▶ Development (DEV): The sandbox where the developers work and perform internal testing and development.
- ▶ User Acceptance Testing (UAT): This is where the developers promote their code for user tests to sign off and approve changes or new applications.

- ▶ Production (PRD): This environment is where the day-to-day activities of the business are handled.

Each environment has a primary purpose, as indicated by its respective description in the preceding list. With the latest advances in IBM POWER® technology and operating system virtualization, IBM has made the process of building LPARs much easier and considerably less expensive than in the POWER5 era, so there is truly little reason not to isolate. It might take some time and a call to some third-party vendors. However, there are many supporting reasons for each of these environments to be isolated on separate LPARs. Here are some of the reasons:

- ▶ Development:
 - Sandbox where developers can coordinate internal testing.
 - Separate LPARs offer opportunities to isolate test workload for benchmarking.
 - Stress testing does not adversely affect production workload.
 - Developers have the highest level of control and access.
 - Developers can work with sanitized data to ensure that security policies are maintained.
- ▶ User Acceptance Testing:
 - Sterile environment for user testing.
 - Receives changes through change management control.
 - Isolated for protection of the production environment.
 - Administrators have control, but developers can also adjust things.
- ▶ Production:
 - Where the users live and work.
 - Receive changes through change management control.
 - Administrators control and developers are locked out.

12.1.8 What is Zend Server

Zend Server is a supported and tested PHP server that comes with IBM i. It installs as a licensed program and receives updates through a PTF process much like any other IBM licensed program. The server is shipped on IBM i installation media, but is likely not the most current version. Because the PHP community makes updates on a more regular schedule, the physical media is difficult to keep current. The most current version of the Zend Server software and updates are available at <http://www.zend.com>, which is where you should get your PHP run time.

Because PHP is a server-centric language, the PHP code is stored and run on the IBM i in an environment that is known as IBM Portable Application Solutions Environment for i (PASE for i). The PASE environment, which is essentially a light-weight IBM AIX run time, allows UNIX binary files to run in the native IBM i environment. Because many of the components of the PHP “stack” are built in a UNIX landscape, the PASE environment provides the fastest path to development for IBM i while still allowing the typical “green screen” command processing to manage the start and stop of the server.

PHP's whole purpose is to produce something the browser can process. Browsers can process nearly everything in the web world, including HTML, JavaScript, CSS, XML, and so on. PHP runs on the server with a mission to send things to the browser and react to things coming back. PHP is essentially to the web what RPG is to the 5250 interface.

12.2 PHP development

Because PHP as a language was developed for web pages, it is a perfect fit for server-centric application development in web browsers. This development requires a new set of skills and tools that initially can be learned quickly and then grown over time. It is important to develop HTML, CSS, JavaScript, and other web skills, and these skills can be developed while your skills in PHP grow. Having a good integrated development environment (IDE) is also important because an IDE can support your PHP applications from beginning to end. There are many good IDEs that support PHP. Because Zend Studio is provided at no additional charge to IBM i customers for the first year, we explore it in greater detail.

12.2.1 Zend Studio

Zend Studio is a premier IDE for PHP development and is available to IBM i customers at no additional charge for the first year. Support for the first year is covered under an IBM i centric license that is due to the partnership between Zend and IBM. Zend Studio is built on the Eclipse open source project, much like the Rational products. The initial download and installation of the software, available from the Zend website, loads a perpetual license of Zend Server. Licensing Zend Server is a fairly simple process and is outlined in the later in this section.

Installing and starting Zend Studio

Documentation about the installation of Zend Studio is available at the Zend website, but many of the default values suffice for the typical installation.

Clicking the link for a license

When Zend Studio starts for the first time, it prompts you for a license key. The words in the dialog box “Get a license for IBM i” are a hyperlink. Click this link to be taken to the Studio registration page (Figure 12-4 on page 553). Your default browser should open and take you to the appropriate Zend page to complete the registration process.

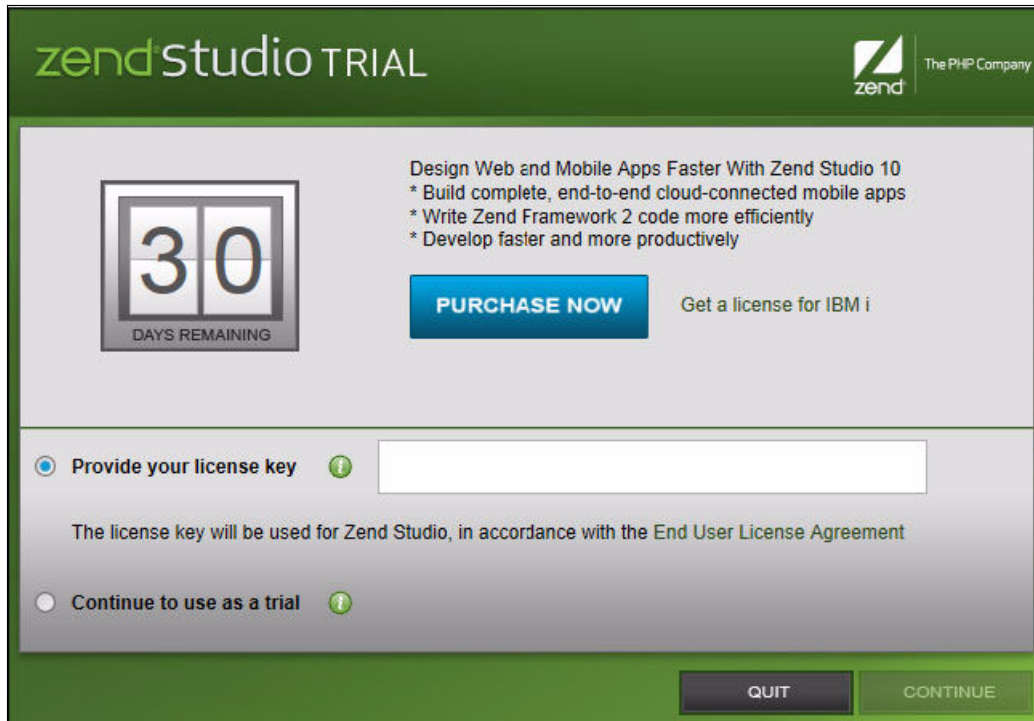


Figure 12-4 License key window

Completing the registration information

You must create and log in with a user account at Zend and have your IBM i system serial number to input into the form (Figure 12-5). Clicking **Complete processing** starts the entitlement process. If you have any questions, ask Zend Technical support through the case creation process at <http://www.zend.com>.

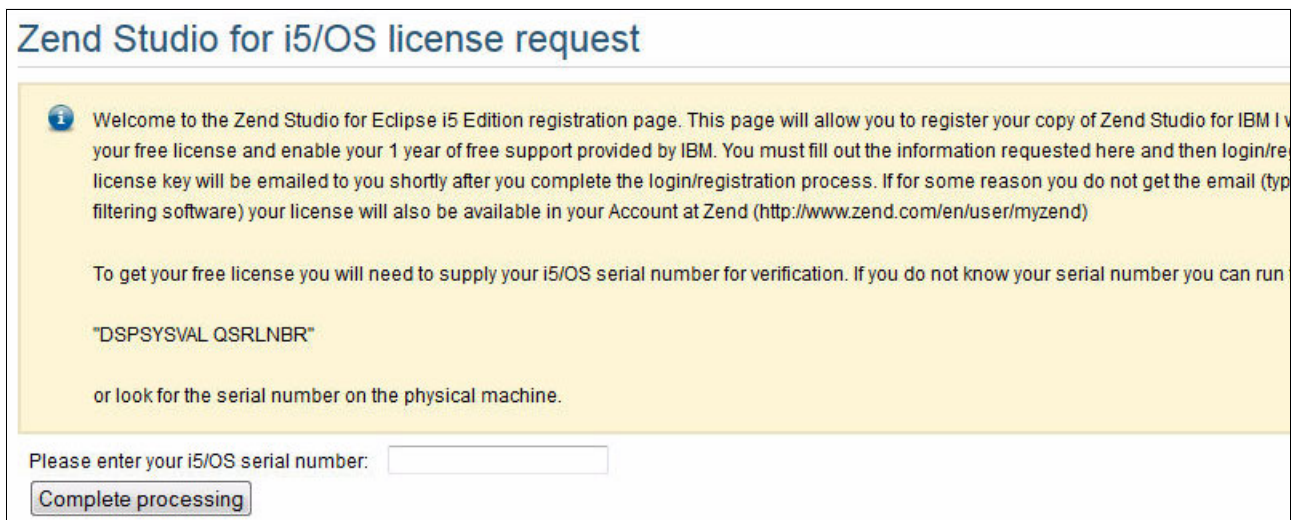


Figure 12-5 Registration information

Applying the license key

A return email come from Zend. This email contains the license key. Copy and paste that key into the window that is shown in Figure 12-5 on page 553 to activate the Zend Studio license. When it is installed, the developer creates a remote connection by using the steps that are described in the following sections.

Setting up a remote connection

A remote connection provides Zend Studio users access to the root file system of your IBM i. It is here that the PHP scripts are stored and retrieved by the PHP stack. To create the remote connection, click **Window** → **Show View** → **Remote Systems** (Figure 12-6).

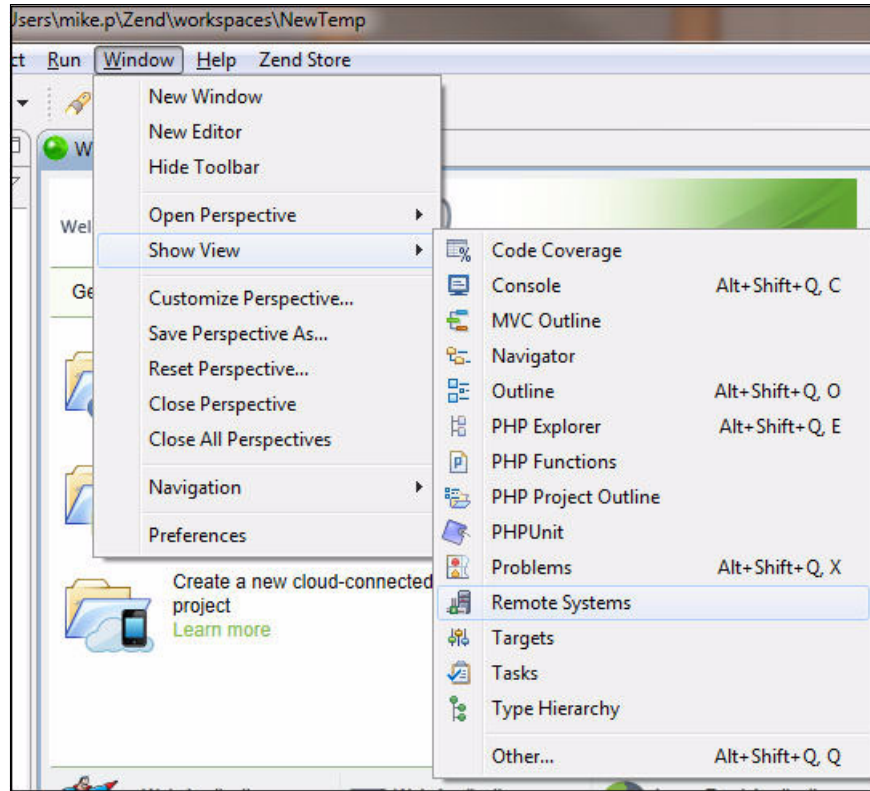


Figure 12-6 Setup remote connection

The remote systems window opens (Figure 12-7). From here, click the intersection icon or right-click and select **New Connection**.

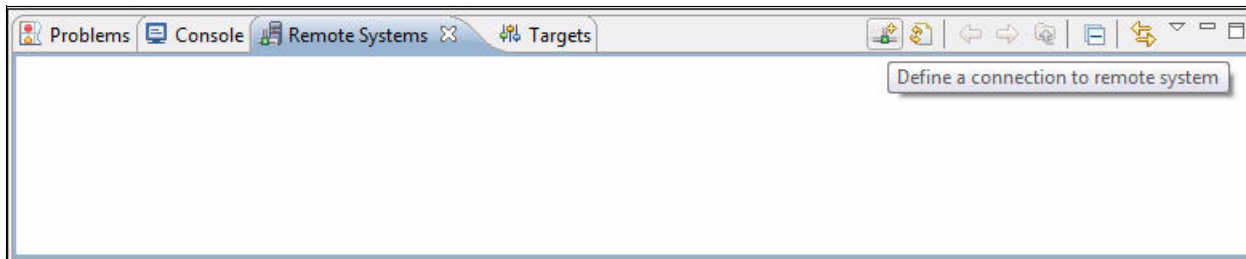


Figure 12-7 New connection

Most Zend Studio users use the SSH function. Select the **SSH Only** check box and click **Next** (Figure 12-8).

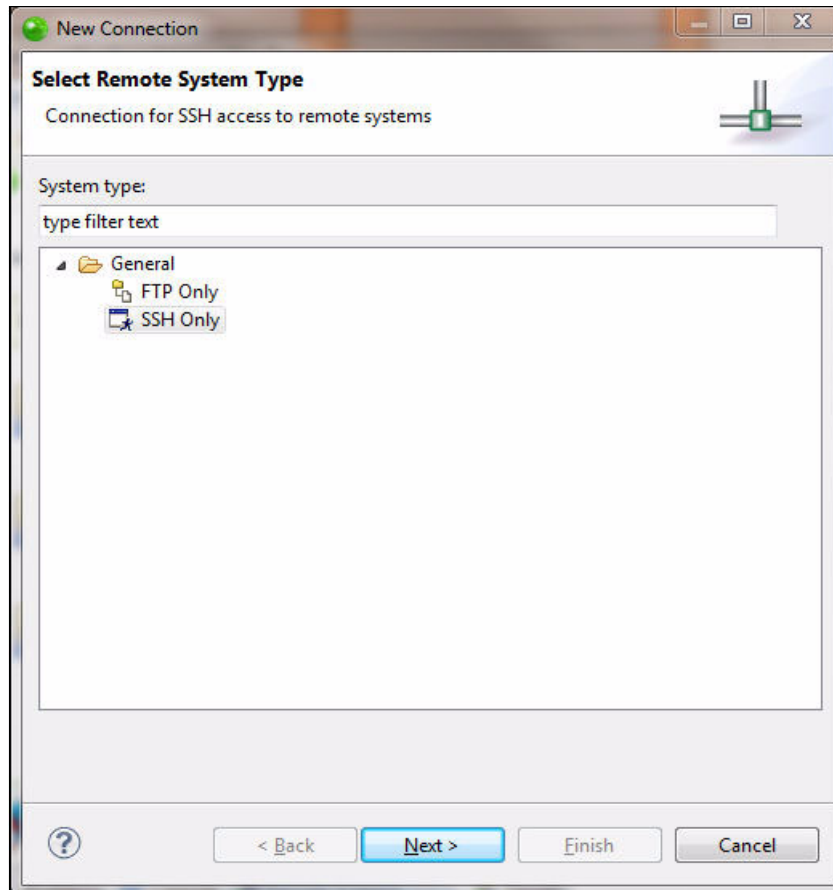


Figure 12-8 Connection for SSH access

Enter the host name or IP address of the server. The connection name and description can be optionally completed and are used only for labeling, as shown in Figure 12-9.

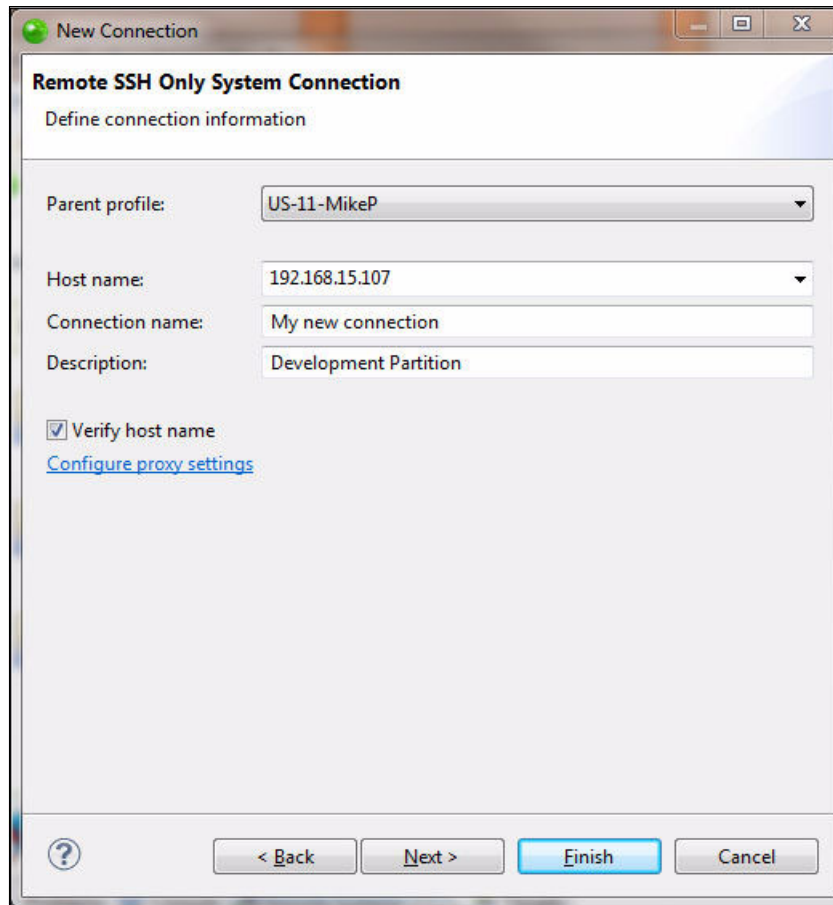


Figure 12-9 Defining a host name

Each of the windows on the Zend Studio desktop can be moved around. Grab the tab and drag it around the desktop and you get a highlight box showing where the box “snaps in”. The window can be snapped in over the PHP Explorer window, as shown in Figure 12-10. Now it appears in a location that is familiar to Rational users.

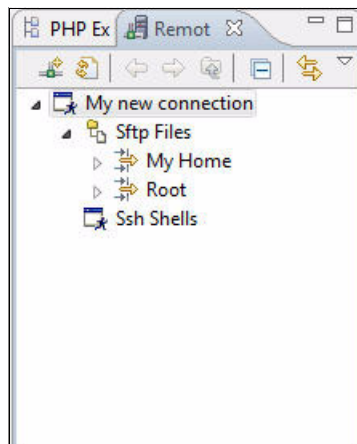


Figure 12-10 PHP Explorer

Now you have a connection and it is time to see whether it works. Click the twistie (triangle) next to Root to be prompted for your IBM i credentials, as shown in Figure 12-11.

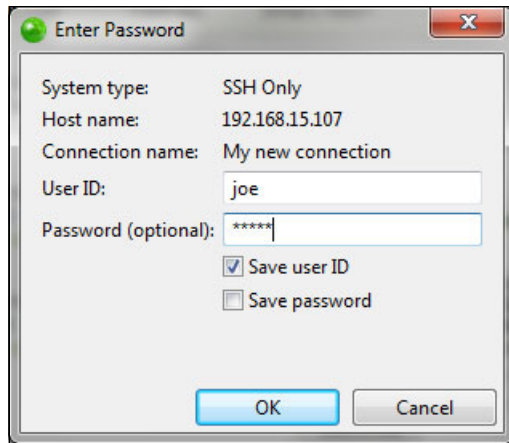


Figure 12-11 Credentials - sign-on information

After your credentials are validated, expand the twistie to get to your directory of choice, as shown in Figure 12-12.

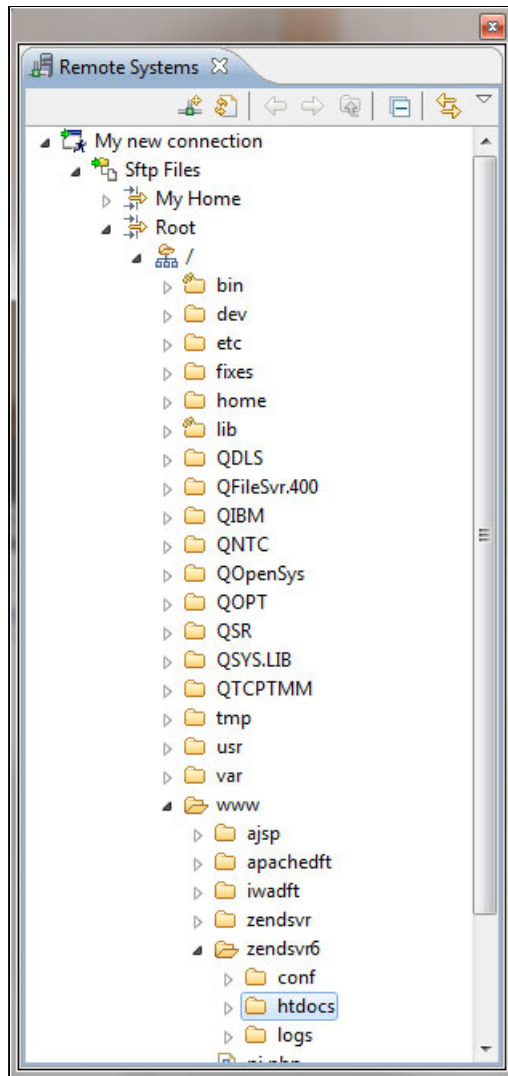


Figure 12-12 Directory structure

Setting up a remote project

Before you get the debugger working, it is important to set up a remote project. This section outlines the required steps.

Programmer Development Tools

Programmer Development Tools (PDT) is an open source project on the <http://www.eclipse.org> site that provides a PHP development environment for Eclipse. Users of Zend Studio receive a complete Eclipse bench, but Rational users might prefer to plug PDT into their bench. This can be achieved by downloading and installing the PDT plug-in from the Eclipse repository into your Rational Developer for i environment.

12.2.2 Exploring the stair-step approach: Four steps

This section explains a step-by-step approach to coding PHP for IBM i.

Inline code

New PHP developers typically journey through a transition of development starting with inline code that is vaguely reminiscent of RPG II style programming. This style is effective but difficult to maintain. It is still the best way to get a fast result and start growing PHP skills. The code sample in Example 12-1 is shipped in the Sample section of Zend Server for IBM i. It illustrates a technique for reading and displaying DB2 data through a browser.

Example 12-1 Displaying DB2 data through a browser

```
<html>
<head>
<title></title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>

<?php

$conn_resource = db2_connect("*LOCAL", "", "");

if (!$conn_resource) {
    echo "Connection failed. SQL Err:";
    echo db2_conn_error();
    echo "<br>";
    echo db2_conn_errormsg();

exit();
}

echo "Executing SQL statement: SELECT * FROM ZENDSVR.SP_CUST WHERE CUST_ID > 1220
FOR FETCH ONLY". "<br>", "<br>";

/* Construct the SQL statement */
$sql = "SELECT * FROM ZENDSVR.SP_CUST WHERE CUST_ID > ? FOR FETCH ONLY";

/* Prepare, bind and execute the DB2 SQL statement */
$stmt= db2_prepare($conn_resource, $sql);
$lower_limit = 1220; //from the CUST_ID value
$fields = db2_num_fields($stmt);
if($fields > 0 )
{
//show Table Header (analyze result set)
    echo "<table border=1>";
    echo "<tr>";
    for($i=0; $i<$fields; $i++)
    {
        echo '<td width="20%">';
        $name = db2_field_name($stmt, $i);
        echo $name;
        echo "</td>";
    }

    echo "</tr>";

//Execute statement , uses a binding of parameters
db2_bind_param($stmt, 1, "lower_limit", DB2_PARAM_IN);
```

```

$result = db2_execute($stmt);

    if (!$result) {
        echo 'The db2 execute failed. ';
        echo 'SQLSTATE value: ' . db2_stmt_error();
        echo ' Message: ' . db2_stmt_errormsg();
    }
    else
    {
        while ($row = db2_fetch_array($stmt))
        {
            echo "<tr>";
            for($i=0; $i<$flds; $i++){
                echo '<td width="20%">';
                echo $row[$i] . '&nbsp;';
                echo "</td>";
            }
            echo "</tr>";
        }
    }
    echo "</table>";
}
?>

```

In Example 12-1 on page 559, you can see all the major components that are required to retrieve data from DB2 and display it through HTML. This is an efficient way to run PHP scripts. However, it is an inefficient way to create and maintain them because there is no effort that is made toward basic concepts such as reuse and structure. The code simply goes where it wants. However, this approach works when you are learning.

Procedural

If you take the code in Example 12-1 on page 559 and structure it more effectively, there are a couple of advantages. The example in Example 12-2 shows a different code set that displays the same output but with functions. In PHP, functions are like subprocedures in RPG. They can be built and assembled at execution time, have local variables, and can allow your application set to become a box of tools. In this crude example, there are a couple of these advantages being built out. First, in this example, we organized the code by logical functions, which are shown in Example 12-2.

Example 12-2 Procedural approach to displaying data

```

<?php

// Procedural approach to displaying data...
function getDB2Connection() {

    // Create DB2 Connection and return to caller...
    $conn_resource = db2_connect ( "*LOCAL", "", "" );

    if (! $conn_resource) {
        echo "Connection failed. SQL Err:";
        echo db2_conn_error ();
        echo "<br>";
        echo db2_conn_errormsg ();
    }
}

```

```

        exit ();
    }

    return $conn_resource;
}
function getData($conn_resource) {

    // Take Database connection and run SQL Statement against it...
    echo "Executing SQL statement: SELECT * FROM ZENDSVR.SP_CUST WHERE CUST_ID >
1220 FOR FETCH ONLY" . "<br>", "<br>";

    /* Construct the SQL statement */
    $sql = "SELECT * FROM ZENDSVR.SP_CUST WHERE CUST_ID > ? FOR FETCH ONLY";

    /* Prepare, bind and execute the DB2 SQL statement */
    $stmt = db2_prepare ( $conn_resource, $sql );
    $lower_limit = 1220; // from the CUST_ID value

    // Execute statement , uses a binding of parameters
    db2_bind_param ( $stmt, 1, "lower_limit", DB2_PARAM_IN );
    $result = db2_execute ( $stmt );
    if (! $result) {
        echo 'The db2 execute failed. ';
        echo 'SQLSTATE value: ' . db2_stmt_error ();
        echo ' Message: ' . db2_stmt_errormsg ();
    }

    return $stmt;
}
function displayData($stmt) {
    echo '<html><head><title>Procedural Example</title>';
    echo '<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">';
    echo '</head><body>';

    $flds = db2_num_fields ( $stmt );
    if ($flds > 0) {
        // show Table Header (analyze result set)
        echo "<table border=1>";
        echo "<tr>";
        for($i = 0; $i < $flds; $i ++) {
            echo '<td width="20%">';
            $name = db2_field_name ( $stmt, $i );
            echo $name;
            echo "</td>";
        }

        echo "</tr>";

        while ( $row = db2_fetch_array ( $stmt ) ) {
            echo "<tr>";
            for($i = 0; $i < $flds; $i ++) {
                echo '<td width="20%">';
                echo $row [$i] . '&nbsp;';
                echo "</td>";
            }
        }
    }
}

```

```

        }
        echo "</tr>";
    }
}
echo "</table></body></html>";
}

$connect = getDB2Connection ();
$resultSet = getData ( $connect );
$display = displayData ( $resultSet );

?>

```

The value of the procedural approach to PHP development becomes evident in several ways. First, all the variables that are used in the functions become local to that function. Therefore, the debugging of problems in a PHP script becomes much easier because there are no more global variables. Anyone who has made the leap from RPG III to subprocedures has experienced the benefit of debugging 150 lines of code over a 5000 line program with subroutines and global variables.

In addition to local variables, consider that the logic is broken up. So, much like subroutines and subprocedures, we have logically dissected the script so that some pieces can be reused. It seems reasonable that the `getDB2Connection()` function can be used in many more scripts, possibly every script on the system. This code reuse makes the procedural model valuable and a smart investment of time for RPG and COBOL developers who want to implement PHP applications.

Object-oriented PHP

The object model of object-oriented PHP is powerful, and it is optional. This feature of PHP allows RPG and COBOL developers to grow into the OO model of PHP.

Building upon the previous example, you can see things are a little different. The example (Example 12-3) is now two scripts instead of one. However, the “require-once” acts as a “copybook” and brings the PHP code for the class. It becomes much like a service program because it contains the standard code for displaying a customer. In the next file (Example 12-4 on page 564), you can see the customer being displayed by using a procedural invocation. This blends the two environments so that object-oriented developers and procedural developers can work side by side.

Example 12-3 Object-oriented approach to displaying data

```

<?php

// Object Oriented approach to displaying data...Part 1
class displayCustomers {
    private $customer;

    public function __construct($customer="ALL") {

        if ($customer != "ALL") {
            // Validate customer
            $filterOptions = array("options"=>
                array("min_range"=>0, "max_range"=>99999));

```



```

        if (filter_var($customer, FILTER_VALIDATE_INT, $filterOptions) ===
false) {
            exit("Customer number not in range, please try again");
        }
    }

}

private function getDB2Connection() {

    // Create DB2 Connection and return to caller...
    $conn_resource = db2_connect ( "*LOCAL", "", "" );

    if (! $conn_resource) {
        echo "Connection failed. SQL Err:";
        echo db2_conn_error ();
        echo "<br>";
        echo db2_conn_errormsg ();

        exit ();
    }

    return $conn_resource;
}

private function getData() {
    $conn_resource = $this->getDB2Connection ();

    // Take Database connection and run SQL Statement against it...
    echo "Executing SQL statement: SELECT * FROM ZENDSVR.SP_CUST WHERE CUST_ID
> 1220 FOR FETCH ONLY" . "<br>", "<br>";

    /* Construct the SQL statement */
    $sql = "SELECT * FROM ZENDSVR.SP_CUST WHERE CUST_ID > ? FOR FETCH ONLY";

    /* Prepare, bind and execute the DB2 SQL statement */
    $stmt = db2_prepare ( $conn_resource, $sql );
    $lower_limit = 1220; // from the CUST_ID value

    // Execute statement , uses a binding of parameters
    db2_bind_param ( $stmt, 1, "lower_limit", DB2_PARAM_IN );
    $result = db2_execute ( $stmt );
    if (! $result) {
        echo 'The db2 execute failed. ';
        echo 'SQLSTATE value: ' . db2_stmt_error ();
        echo ' Message: ' . db2_stmt_errormsg ();
    }

    $records = array();
    while ( $row = db2_fetch_assoc ( $stmt ) ) {
        $records[] = $row;
    }

    return $records;
}

public function displayData() {

```

```

$records = $this->getData();

echo '<html><head><title>00 Example</title>';
echo '<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">';
echo '</head><body>';
echo "<table border=1>";
echo "<tr>";

$firstRow = $records[0];
foreach ($firstRow as $fieldName=>$fieldValue) {
    // show Table Header (analyze result set)
    echo '<td width="20%">';
    echo $fieldName;
    echo "</td>";
}

echo "</tr>";

// Now print rows...
foreach ($records as $record) {
    echo "<tr>";
    foreach ($record as $field) {
        echo '<td width="20%">';
        echo $field . '&nbsp;';
        echo "</td>";
    }
    echo "</tr>";
}
echo "</table></body></html>";
}
?>

```

Example 12-4 shows a procedural display call.

Example 12-4 Procedural display call

```

<?php

require_once 'db2sqlx3.php';

$dspCustObj = new displayCustomers();

$dspCustObj->displayData();

?>

```

Frameworks

As you can see, you must add quite a bit of code for a simple example. This process is similar to complex RPG programs where you successfully implement subprocedures. PHP scripts can and should use many small, trusted routines. Frameworks are nothing more than code libraries of trusted routines. Some of the more popular frameworks include CAKE, Symfony, and Zend Framework. Zend Server comes preinstalled with Zend Framework 1 and 2 for your convenience. However, developers are not limited to these frameworks because the PHP stack on IBM i supports any framework.

Most frameworks are open source projects and, therefore, developed by community members. In many cases, these community members share a passion for developing solutions that are smart and can scale effectively. The community around Zend Framework is led by developers at Zend Technologies, but enjoys the contributions of hundreds of developers from around the world.

We are going to take our example again and now implement it using a typical Zend Framework 2 approach.

12.2.3 Other data access

In addition to DB2, PHP running on IBM i has drivers to support access to MySQL, Microsoft SQL Server, Oracle, SQL Lite, and more. This section focuses on MySQL because it can also run natively on IBM i and has many versatile purposes for running alongside DB2.

MySQL

Because DB2 works so well, there might not be an obvious reason to be interested in another database. PHP was born in the world of the LAMP stack, that is, Linux, Apache, MySQL, and PHP. All of these components are open source projects and, as a result, require no financial investment to get started and to keep running. This is attractive to independent consultants and “not-for-profit” entities, where the entire staff is volunteers and there is absolutely no funding for investing in systems. A \$10 a month website and these tools can produce wonderful church and volunteer group websites in minutes.

That still does not answer why an IBM i developer might be interested in MySQL. The reality is that many of the open source website building tools have become so wildly popular and powerful that they have become useful and trusted in the enterprise. Projects such as SugarCRM, Magento Commerce, Wordpress, and Drupal are examples. The developers of these projects built them primarily with MySQL in mind. So, it makes sense that providing MySQL to the IBM i community can mean instant access to thousands of open source (or even commercial) applications that can be integrated into your IBM i enterprise to enhance your RPG applications.

Many organizations, such as “COMMON, A Users Group”, use PHP and an open source project like Joomla to run their website. This application runs natively on IBM i along with many of their other applications. Data access using MySQL on IBM i is no different than any other server. Therefore, any open source application should install and work on IBM i. The people who run <http://www.youngipprofessionals.com> have many examples of PHP open source applications running natively on IBM i. They even have instructions about how to help you set up applications on your system.

MySQL is implemented on IBM i through a no charge product called ZendDBi. This product is yet another benefit to the IBM i community as a result of the partnership between Zend and IBM. IBM i customers can install MySQL optionally as part of Zend Server or separately by downloading ZendDBi from the Zend website.

Command-line access to MySQL is available through the “green screen” terminal emulation for PASE in either Qshell or QP2TERM. Most people choose to install an open source application such as phpMyAdmin or Adminer to use a GUI interface to administer MySQL natively on IBM i.

One fundamental benefit to the implementation of ZendDBi is that it ships with a copy of the DB2 Storage Engine. The architecture of MySQL is federated and allows different simultaneous data stores across the server infrastructure. The default data store for MySQL 5.1 is MyISAM. This is a great data store for content management systems that require single-tenant access because the tables are typically locked for writes to the database. For multi-tenant database access, MySQL recommends the InnoDB storage engine with its record locking and full atomicity. This is the default for MySQL 5.5 and higher.

The DB2 Storage Engine is special because it allows the administrator to create tables in MySQL while the data is stored in DB2. This feature allows an application accessing MySQL to use all the MySQL functions and still have the data in DB2. As a specific point, these tables are not replicated and the data is stored in only one place, DB2. This gives you the best of both database types. Your open source applications work with no changes and all your data is in a single DB environment, DB2.

12.3 HLL integration

Using existing business logic is essential to many modernization strategies. “If it works, do not fix it” is a phrase that comes to mind. However, some applications can be modernized from the RPG perspective and that has value. PHP modernization can make great use of API-like HLL language programs like RPG, COBOL, and CL. The program object type ensures that each program created has a consistent interface and, as a result, can be called by PHP using the open source toolkit. This toolkit also allows access to native IBM i artifacts like data areas, data queues, system values, spooled files, and more. As of this writing, all of the source code for the toolkit is stored and available at the Young i Professionals website, found at:

<http://www.youngiprofessionals.com>

12.3.1 What the toolkit is made of

The open source toolkit for PHP is made up of two parts. The first part is XMLSERVICE and this part of the toolkit is developed and maintained by IBM. The XMLSERVICE source code is essentially ILE RPG, DB2 Stored Procedures, and CL. The selection of these languages was made by IBM because the intention was to develop an open source project using common native IBM i language. Because most of the developers on IBM i are RPG developers, writing the native side of the toolkit in RPG made sense because it allows RPG developers to participate in the maintenance and new enhancements of the toolkit.

The second part of the toolkit is written in PHP classes and implemented through objects in PHP. The object-oriented model allows for continuous improvement of the toolkit because classes are implemented at run time instead of when the system is built. Customers and volunteers can modify the toolkit elements on both sides. The PHP classes use the XMLSERVICE when accessing IBM i native objects.

12.3.2 High-level description and workflow

The interface between the two sides of the toolkit is an XML payload. Essentially, the PHP toolkit receives a request from a PHP script to call a program. The PHP classes wrap up the request and package it in an XML file. The XML file is then transmitted from the PASE environment to the native environment through DB2 Stored Procedures or a shared memory model where the address spaces of the file are sent to XMLSERVICE. Then, the XMLSERVICE functions take over, unwrap the XML file, and implement the call that is requested. When the program call completes, XMLSERVICE takes the response and repackages it in another XML package to send back to PHP along the same path in which it was received initially. Figure 12-13 illustrates this topology and how it works.

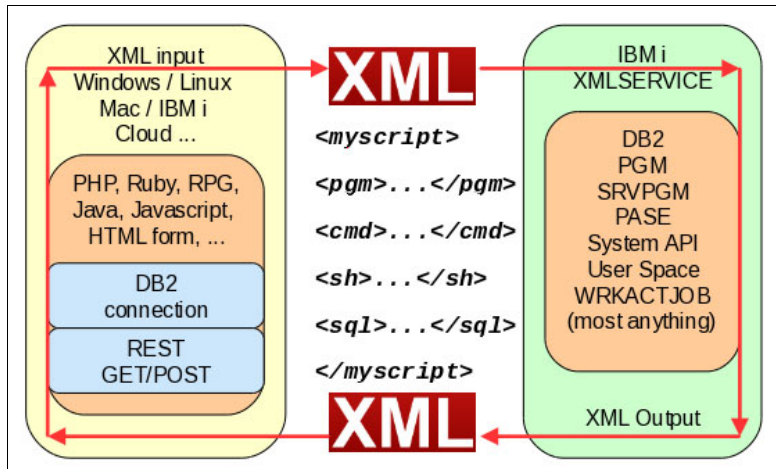


Figure 12-13 Flow of request from PHP through XMLSERVICE to the IBM i and back

As Figure 12-13 indicates, not only PHP but many other languages can use the XMLSERVICE facility to access native IBM i resources. The process is not only repeatable but highly scalable. The only difference is that with PHP there are some native PHP classes to help interface between the core PHP application and the XMLSERVICE component.

12.3.3 Program calls

Program calls are the first place IBM i developers typically like to start with the toolkit because they represent the most value for the time investment. Think of a pricing program in an enterprise resource planning (ERP) system. In many ERP systems, the pricing program is made up of hundreds or even thousands of lines of code and might access dozens of DB2 tables, possibly even calling other RPG programs. To rewrite this pricing program in another language is inefficient on two levels: it is an unnecessary duplication of effort, and there is a high probability that the new program does not calculate a price the exact same way. Calling the existing, tried, and trusted pricing program in RPG, for example, makes much more sense than rewriting and performing expensive parallel maintenance. In our example, we start with a simple RPG program and gradually increase the complexity until we have used most of the popular examples.

Simple RPG call

It is a good idea to start working with the open source toolkit using a simple RPG program. Think of how you might have learned subfiles. First, you start with a load all, then a page-at-a-time subfile, and, finally, a page-by-page subfile. This process allows you to gain confidence as your skills grow, and the same approach here helps you absorb the material better.

The first example is some simple business logic in RPG that performs a standard line discount. This line discount is based on simple quantities. The PHP method accepts two parameters and returns an adjusted price. To the overall PHP application, this looks like a black box. Opening the functions reveals much more. The line discount program does not access any tables. It calculates a discount of 20% for a quantity over 75, 15% for a quantity 50-75, 10% for a quantity 25 - 50, and no discount for quantities under 25. This is handled easily in an RPG program, as illustrated in Example 12-5.

Example 12-5 Sample RPG call

```

<html>
<head>
<TITLE>Order Detail</TITLE>
</head>
<body></body>

<?php
function calcDiscount($discountQuantity, $Price) {

    // Setup includes for toolkit classes...
    require_once 'ToolkitService.php';

    static $PHPToolkitObj;

    if(!isset($_SESSION))
    {
        $_SESSION['sessid'] = session_id();
    }
    $session = $_SESSION['sessid'];

    if (! is_object ( $PHPToolkitObj )) {
        // Set the authorization values... IBM i credentials...
        $db = 'SALES1';
        $user = '';
        $pass = '';
        $extension = 'ibm_db2';
        static $PHPToolkitObj;

        // Using try & catch, instantiate the toolkit object...
        if (! is_object ( $PHPToolkitObj )) {
            try {
                $PHPToolkitObj = ToolkitService::getInstance ( $db, $user, $pass,
$extension );
            } catch ( Exception $e ) {
                echo $e->getMessage (), "\n";
                exit ();
            }
            $PHPToolkitObj->setToolkitServiceParams ( array (
                'InternalKey' => "/tmp/$session" ) );
        }
    }

    // Set up parameters...

```

```

    $qty = $discountQuantity; $pricein = $Price;$priceout = 0.0;$date = '';

    $param [] = $PHPToolkitObj->AddParameterPackDec ( 'both', 9, 0, 'QUANTITY',
'QUANTITY', $qty );
    $param [] = $PHPToolkitObj->AddParameterPackDec ( 'both', 13, 2, 'PRICEIN',
'PRICEIN', $pricein );
    $param [] = $PHPToolkitObj->AddParameterPackDec ( 'both', 13, 2, 'PRICEOUT',
'PRICEOUT', $priceout );
    $param [] = $PHPToolkitObj->AddParameterChar ( 'both', 10, 'DATE', 'DATE',
$date );

    // Call the RPG program...
    $result = $PHPToolkitObj->PgmCall ( "RPGPRICE", "ZENDDATA", $param, null, null
);

    // Print the results...
    if (isset ( $result ['io_param'] )) {
        $output = $result ['io_param'];
        $priceout = $output ['PRICEOUT'];
        $date = $output ['DATE'];
    }
    // echo '</pre>'; print_r($result); echo '</pre>'; exit();
    return array ( $priceout, $date );
}

// Main PHP script...

echo '<table border="1"><tr><th>Price</th><th>Quantity</th><th>Discount
Price</th>' . '<th>Date</th></tr>';

$quantityArray = [ 27, 35, 56, 77 ];

$price = 50;

/* Foreach loop to spin through quantity array... */
foreach ( $quantityArray as $quantity ) {

    // calculate discountPrice, here:
    $discountPrice = calcDiscount ( $quantity, $price );

    // Print Values...
    print "<tr><td>" . number_format ( $price ) . "</td>";
    print "<td>" . number_format ( $quantity ) . "</td>";
    print "<td>$" . number_format ( $discountPrice ['0'] ) . "</td>";
    print "<td>" . date ( 'M d, Y', strtotime ( $discountPrice ['1'] ) ) .
"</td></tr>";
}

echo '</table>';
?></body>
</html>

```

The corresponding script is shown in Example 12-6.

Example 12-6 Sample script example

```
<html>
<head>
<TITLE>Order Detail</TITLE>
</head>
<body></body>

<?php
function calcDiscount($discountQuantity, $Price) {

    // Setup includes for toolkit classes...
    require_once 'ToolkitService.php';

    static $PHPToolkitObj;

    if(!isset($_SESSION))
    {
        $_SESSION['sessid'] = session_id();
    }
    $session = $_SESSION['sessid'];

    if (! is_object ( $PHPToolkitObj )) {
        // Set the authorization values... IBM i credentials...
        $db = 'SALES1';
        $user = '';
        $pass = '';
        $extension = 'ibm_db2';
        static $PHPToolkitObj;

        // Using try & catch, instantiate the toolkit object...
        if (! is_object ( $PHPToolkitObj )) {
            try {
                $PHPToolkitObj = ToolkitService::getInstance ( $db, $user, $pass,
$extension );
            } catch ( Exception $e ) {
                echo $e->getMessage (), "\n";
                exit ();
            }
            $PHPToolkitObj->setToolkitServiceParams ( array (
                'InternalKey' => "/tmp/$session" ) );
        }
    }

    // Set up parameters...

    $qty = $discountQuantity; $pricein = $Price;$priceout = 0.0;$date = '';

    $param [] = $PHPToolkitObj->AddParameterPackDec ( 'both', 9, 0, 'QUANTITY',
'QUANTITY', $qty );
    $param [] = $PHPToolkitObj->AddParameterPackDec ( 'both', 13, 2, 'PRICEIN',
'PRICEIN', $pricein );
```



```

    $param [] = $PHPToolkitObj->AddParameterPackDec ( 'both', 13, 2, 'PRICEOUT',
'PRICEOUT', $priceout );
    $param [] = $PHPToolkitObj->AddParameterChar ( 'both', 10, 'DATE', 'DATE',
$date );

    // Call the RPG program...
    $result = $PHPToolkitObj->PgmCall ( "RPGPRICE", "ZENDDATA", $param, null, null
);

    // Print the results...
    if (isset ( $result ['io_param'] )) {
        $output = $result ['io_param'];
        $priceout = $output ['PRICEOUT'];
        $date = $output ['DATE'];
    }
    // echo '</pre>'; print_r($result); echo '</pre>'; exit();
    return array ( $priceout, $date );
}

// Main PHP script...

echo '<table border="1"><tr><th>Price</th><th>Quantity</th><th>Discount
Price</th>' . '<th>Date</th></tr>';

$quantityArray = [ 27, 35, 56, 77 ];

$price = 50;

/* Foreach loop to spin through quantity array... */
foreach ( $quantityArray as $quantity ) {

    // calculate discountPrice, here:
    $discountPrice = calcDiscount ( $quantity, $price );

    // Print Values...
    print "<tr><td>" . number_format ( $price ) . "</td>";
    print "<td>" . number_format ( $quantity ) . "</td>";
    print "<td>$" . number_format ( $discountPrice ['0'] ) . "</td>";
    print "<td>" . date ( 'M d, Y', strtotime ( $discountPrice ['1'] ) ) .
"</td></tr>";
}

echo '</table>';
?></body>
</html>

```

We look at the script a part at a time. Example 12-6 on page 570 is composed of a single PHP function and a main section of code. This can easily be made into an OO example, but we want to illustrate the ability of the toolkit to coexist with both procedural and OO PHP code.

The function in the PHP code accepts two parameters: price and quantity. These parameters are supplied to the program call to receive the discounted price and date of the transaction. The first line of code inside the function definition is a “require_once” that is a “copybook” function that goes out to find the source code for the class and inserts it in the script. This source code is stored in a directory that is defined on the “include_path” of the php.ini file and this value is maintained easily through the Zend Server admin interface. At script execution time, the include path is searched for the file much like the library list can be searched for an object.

Next, you see several variables being defined. One is defined as static. Each variable in a function is destroyed when the function ends. Static variables, however, persist from function call to function call. This eliminates the need to re-create the object every time the function is called and thus improves performance.

We check for session state and set it if it is not turned on. Sessions are a way to allow a single user/browser a unique identifier so that PHP can perform the illusion of statefulness. By default, web pages are stateless, meaning that the server really does not track activities from one page call to another. Session state allows a PHP developer to provide the user with the perception of statefulness, much like you see when you go to an online shopping site and you add things to your cart. These values are often stored in session variables for quick and easy access. We start a session here because we are going to call the RPG program several times and we want good performance.

Library lists

Usage of the library list has been part of the IBM i space for decades. Calling programs requires a little extra care from outside the native environment and there are various ways to use these features. Example 12-7 explores two such features: a command call and a program call requiring a database file access. There are other ways to use library lists with user credentials and other items. Those methods are not described here, but can be found in the toolkit documentation.

Example 12-7 Command call and program call

```
<?php

// Bring in the toolkit...
require_once 'ToolkitService.php';

function getSessions() {

    // Check session state, if active, return ID else create session...
    if (session_status == PHP_SESSION_ACTIVE)
        $sess = session_id ();
    else {
        session_start ();
        $sess = session_id ();
    }
    return $sess;
}

$sess = getSessions ();
// Call the open source toolkit for product info...

$db = 'SALES1';
$user = '';
$pass = '';
```

```

$extension = 'ibm_db2';
$ohQty = 0;
$error = '';
$prod = 2;

$ToolkitServiceObj = ToolkitService::getInstance ( $db, $user, $pass, $extension
);
$ToolkitServiceObj->setToolkitServiceParams ( array (
    'InternalKey' => "/tmp/$sess"
) );

$param [] = $ToolkitServiceObj->AddParameterPackDec ( 'both', 9, 0, 'PRODUCT',
'PRODUCT', $prod );
$param [] = $ToolkitServiceObj->AddParameterPackDec ( 'both', 10, 0, 'ONHQTY',
'ONHQTY', $ohQty );
$param [] = $ToolkitServiceObj->AddParameterChar ( 'both', 256, 'ERROR', 'ERROR',
$error );
$result = $ToolkitServiceObj->CLCommand ( "RMVLIBLE ZENDDATA" );
$result = $ToolkitServiceObj->CLCommand ( "ADDLIBLE ZENDDATA" );
$result = $ToolkitServiceObj->PgmCall ( "PRODAVAIL", "*LIBL", $param, null, null
);

if ($result) {
    print 'On hand quantity for item ' . $prod . ' is ' . $result ['io_param']
['ONHQTY'];
} else

    echo 'Execution failed';

$ToolkitServiceObj->disconnect ();

```

The RPG program is nothing special. In fact, this can easily be handled with an SQL statement. Focus on the mechanics because your product availability routines are more sophisticated and they can easily be implemented the same way. RPG opens a typical PRODINV table and proceeds to chain to a record, retrieve data values, calculate the quantity available, and return.

Focusing on the PHP script shows that we begin the setup of the toolkit call much like Example 12-6 on page 570 and, adding a few new items, starting with the setup of the library list. The **ADDLIBLE CL** command is run using the PHP toolkit, similar to how a CL program might add a library list before a program call. There is a different method call before the program call. These method calls run a CL command to set up the library list so that the program can find the data file for which it is looking. Customers with a single LPAR for both development and production might find this technique useful. In the program call, we switched the library reference from a hardcoded library to that of *LIBL to tell the toolkit that the library of the RPG program and all its referenced objects should be found in the library list.

The output in our example matches the data that is expected (Example 12-8).

Example 12-8 Output display

On hand quantity for item 2 is 5000

12.4 How dates are different in PHP

Dates are essential to any transaction-based system. Storing and retrieving dates is done in nearly every table of an ERP system and the ancillary systems that surround the main business processing of most organizations. Understanding how to work with dates in PHP is essential to mastering these values.

Example 12-9 is a simple PHP script that displays system time.

Example 12-9 Display system time example

```
<?php

$date = date('m/d/Y G:i:s');

echo "Hello World, today is $date";

echo "<br><br>";

$date = date('l, F j, Y');

$time = date('g:i:sa');

echo "Or, if you prefer a more formal format, then today is $date at $time";
?>
```

To make this example work as you might expect on your system, you might need to change the default time zone of GMT to your particular time zone in the `PHP.ini` file. To do this, navigate to the Zend Server Administrative Interface, click the **Components** tab and the **PHP** subtab. Scroll down to and click **date** to expand the view. If your default is GMT and you are not in GMT, enter the current time zone. Click **Save** and restart Zend Server to implement the change because the `PHP.ini` file is interpreted only when PHP starts.

Figure 12-14 on page 575 shows the interface for setting your time zone correctly.

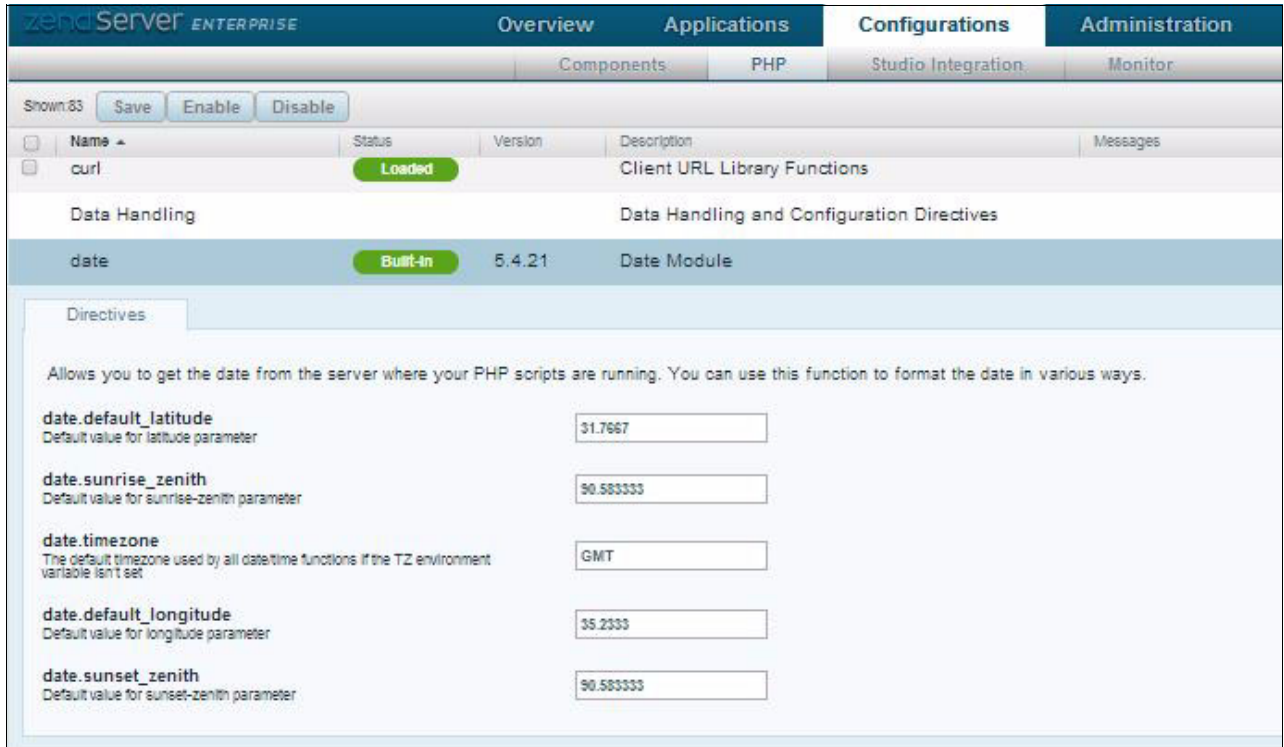


Figure 12-14 PHP runtime file before setting the time zone correctly.

When you have updated the time zone value to your correct local time zone, you must remember to restart the PHP server for this change to work as expected (Figure 12-15).

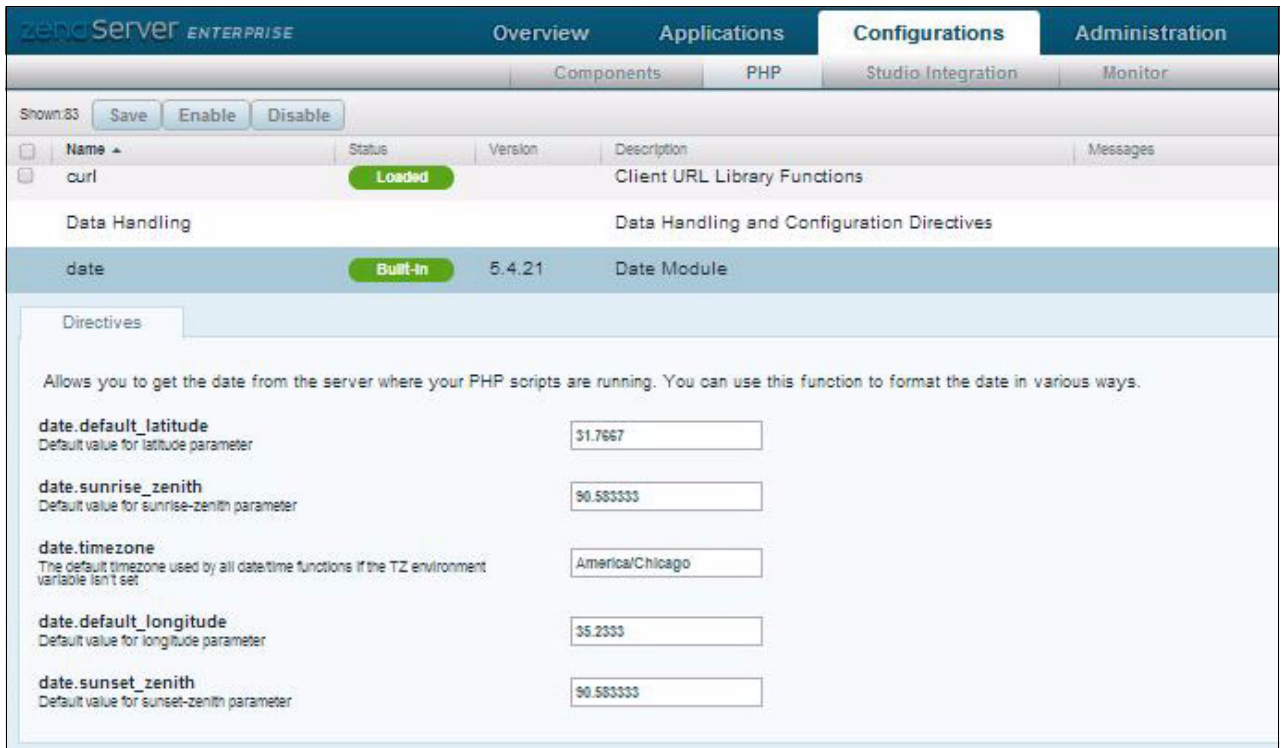


Figure 12-15 After the time zone is set to the correct local value

The output of Example 12-9 on page 574 is shown in Figure 12-16.

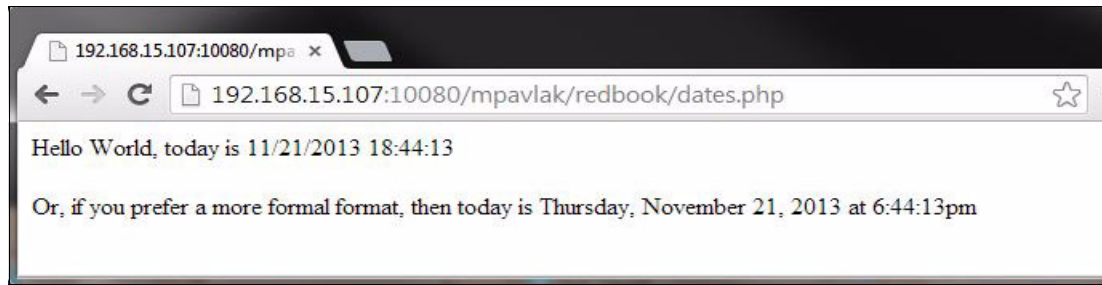
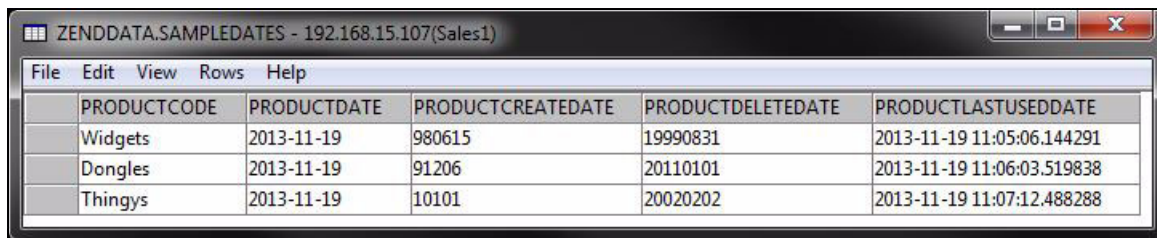


Figure 12-16 Output of the script

Notice the way that the system collects the time. There is a function that is called **date()** that accepts two parameters. In the preceding examples, there is only one parameter that identifies the format of the date to be displayed. The second parameter, the time stamp, is defaulted to the current time stamp when it is not specified. All of the parameters and options within the parameters for functions such as **Date()** are outlined at <http://www.Php.net>. Try to avoid memorizing the nearly 5000 functions that make up the language. Instead, focus on how to find these functions at the website.

It is important to understand that while DB2 and RPG support date fields, PHP does not. PHP supports only four scalar data types: string, Integer, float, and Boolean. Of those types, when you read a date field or numeric field from a DB2 table, you receive either an integer or a string because Boolean is true/false fields and floats are created if you have decimal precision. In the following sample table (Figure 12-17), you can see various typical dates that might be found in an RPG application. Not all dates are converted to date fields in RPG. Being able to compensate for this is essential. The script that follows does a good job of taking some dates that are illustrated in this table and converting them to UNIX time stamps. When the date is in time stamp form, it can be fed to the **date()** function to be displayed in a human-readable format, as shown in Figure 12-17.



PRODUCTCODE	PRODUCTDATE	PRODUCTCREATEDATE	PRODUCTDELETEDATE	PRODUCTLASTUSEDDATE
Widgets	2013-11-19	980615	19990831	2013-11-19 11:05:06.144291
Dongles	2013-11-19	91206	20110101	2013-11-19 11:06:03.519838
Thingys	2013-11-19	10101	20020202	2013-11-19 11:07:12.488288

Figure 12-17 Typical date that is found in an RPG program

Looking at the SQL statements that are shown in Figure 12-18 on page 577, you can see that the date fields in the example file are various typical formats that might be found in an RPG business application.

```

CREATE TABLE ZENDDATA.SAMPLEDATES (
    PRODUCTCODE FOR COLUMN PRODCODE CHAR(10) CCSID 37 NOT NULL ,
    PRODUCTDATE FOR COLUMN PRODDATE DATE NOT NULL DEFAULT CURRENT_DATE ,
    PRODUCTCREATEDATE FOR COLUMN PRODCRDT DECIMAL(6, 0) NOT NULL ,
    PRODUCTDELETEDATE FOR COLUMN PRODDLDT DECIMAL(8, 0) NOT NULL ,
    PRODUCTLASTUSEDDEDATE FOR COLUMN PRODLUDT TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP )

RCDfmt SAMPDATE1 ;

LABEL ON TABLE ZENDDATA.SAMPLEDATES
    IS 'Sample Date Table for PHP Redbook' ;

LABEL ON COLUMN ZENDDATA.SAMPLEDATES
( PRODUCTCODE IS 'Product Identifier',
  PRODUCTDATE IS 'Product Identifier',
  PRODUCTCREATEDATE IS 'Product Create Date',
  PRODUCTDELETEDATE IS 'Product Delete Date',
  PRODUCTLASTUSEDDEDATE IS 'Product Last Used' );

LABEL ON COLUMN ZENDDATA.SAMPLEDATES
( PRODUCTCODE TEXT IS 'Product Identifier',
  PRODUCTDATE TEXT IS 'Product Date',
  PRODUCTCREATEDATE TEXT IS 'Product Create Date',
  PRODUCTDELETEDATE TEXT IS 'Product Create Date',
  PRODUCTLASTUSEDDEDATE TEXT IS 'Product Create Date' );

```

Figure 12-18 Example SQL statements

If you dump the contents of one of these rows using the `var_dump()` function in PHP, you see that all of these columns come in as strings (Figure 12-19).

```

array(5) {
  ["PRODUCTCODE"]=>
  string(10) "Widgets  "
  ["PRODUCTDATE"]=>
  string(10) "2013-11-19"
  ["PRODUCTCREATEDATE"]=>
  string(6) "980615"
  ["PRODUCTDELETEDATE"]=>
  string(8) "19990831"
  ["PRODUCTLASTUSEDDEDATE"]=>
  string(26) "2013-11-19-11.05.06.144291"
}

```

Figure 12-19 Displaying rows by using the `var_dump()` function in PHP

What was a date field and is now in string format must become a UNIX time stamp to be interpreted by PHP and formatted by using the `date` function. You might argue that formatting dates in RPG is advantageous, and there is some truth to that thought. However, most readers of this text already know how to do that. This section explores converting with PHP so that the developer can make an informed decision.

Consider the output of the code in Figure 12-20. Each format is in its own column with the raw date on the first row and the PHP formatted date on the second row. Although this information is useful, it is the code samples in Example 12-10 and Example 12-11 on page 580 that are truly valuable.

PRODUCTCODE	PRODUCTDATE	PRODUCTCREATEDATE	PRODUCTDELETEDATE	PRODUCTLASTUSEDDATE
Widgets	2013-11-19	980615	19990831	2013-11-19-11.05.06.144291
	11/19/2013	06/15/1998	08/31/1999	11/19/2013 11:05:06am
Dongles	2013-11-19	91206	20110101	2013-11-19-11.06.03.519838
	11/19/2013	12/06/2009	01/01/2011	11/19/2013 11:06:03am
Thingys	2013-11-19	10101	20020202	2013-11-19-11.07.12.488288
	11/19/2013	01/01/2001	02/02/2002	11/19/2013 11:07:12am

Figure 12-20 Date formatting output

Example 12-10 DB2 SQL example

```
<?php

// Object Oriented approach to displaying data...Part 1
class displayProducts {
    private $product;

    public function __construct() {

    }
    private function getDB2Connection() {

        // Create DB2 Connection and return to caller...
        $conn_resource = db2_connect ( "*LOCAL", "", "" );

        if (! $conn_resource) {
            echo "Connection failed. SQL Err:";
            echo db2_conn_error ();
            echo "<br>";
            echo db2_conn_errormsg ();

            exit ();
        }

        return $conn_resource;
    }
    private function getData() {
        $conn_resource = $this->getDB2Connection ();

        /* Construct the SQL statement */
        $sql = "SELECT * FROM ZENDDATA.SAMPDATES";

        $result = db2_exec ( $conn_resource, $sql );
        if (! $result) {
            echo 'The db2 execute failed. ';
            echo 'SQLSTATE value: ' . db2_stmt_error ();
            echo ' Message: ' . db2_stmt_errormsg ();
        }
    }
}
```



```

$records = array();
while ( $row = db2_fetch_assoc ( $result ) ) {
    $records[] = $row;
    //echo '<pre>'; var_dump($row); echo '</pre>';
}

return $records;
}

public function displayData() {
    $records = $this->getData();

    echo '<html><head><title>Dates Example</title>';
    echo '<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">';
    echo '</head><body>';
    echo "&<table border=1>";
    echo "<tr>";

    $firstRow = $records[0];
    foreach ( $firstRow as $fieldName=>$fieldValue ) {
        // show Table Header (analyze result set)
        echo '<td width="20%">';
        echo $fieldName;
        echo "</td>";
    }

    echo "</tr>";

    // Now print rows...
    foreach ( $records as $record ) {
        echo "<tr>";
        foreach ( $record as $field ) {
            echo '<td width="20%">';
            echo $field . '&nbsp;';
            echo "</td>";
        }
        echo "</tr>";

        echo "<tr>";
        foreach ( $record as $key=>$field ) {
            echo '<td width="20%">';
            //var_dump($key); exit();
            switch ( $key ) {
                case 'PRODUCTCODE':
                    echo '&nbsp;';
                    break;
                case 'PRODUCTDATE':
                    $date = strtotime($field);
                    $date = date('m/d/Y', $date);
                    echo $date.'&nbsp;';
                    break;

                case 'PRODUCTCREATEDATE':

```

```

        if (strlen($field) == 5) $field = '0'.$field;
        $date = substr($field,2,2) . '/' .
            substr($field,4,2) . '/' .
            substr($field,0,2);
        $date = strtotime($date);
        $date = date('m/d/Y', $date);
        echo $date.'&nbsp;';
        break;

    case 'PRODUCTDELETEDATE':
        $date = substr($field,0,4) . '/' .
            substr($field,4,2) . '/' .
            substr($field,6,2);
        $date = strtotime($date);
        $date = date('m/d/Y', $date);
        echo $date.'&nbsp;';
        break;
    case 'PRODUCTLASTUSEDDATE':
        //Convert to SOAP TimeStamp
        $date=substr($field,0,10) . 'T' .
            substr($field,11,2) . ':' .
            substr($field,14,2) . ':' .
            substr($field,17,2);
        $date = strtotime($date);
        $date = date('m/d/Y g:i:sa', $date);
        echo $date.'&nbsp;';
        break;
    }
    echo "</td>";
}
echo "</tr>";
}
echo "</table></body></html>";
}
?>

```

Example 12-11 shows the PHP procedure call.

Example 12-11 PHP procedure call

```

<?php

require_once 'db2sqlx4.php';

$dspProdObj = new displayProducts();

$dspProdObj->displayData();

?>

```

In the display data method, you have four separate approaches to dates that are identified by the extra for each loop with the embedded switch statement. The switch statement might look like an RPG select, but understand that it is not. Usage of the “break” (similar to leave in RPG) is required at the end of the group that is provided, and processing is done when this is reached. It is also important to be aware that fields in the substring are based on the first digit of the count, which is zero and not one, as it is in RPG.

1. The first column, “PRODUCTDATE,” is made up of a Date data type in DB2. Here, you can simply apply the `strtotime()` function to convert the date data type to a UNIX time stamp and feed this to the `date()` function for use.
2. The second column, “PRODUCTCREATEDATE”, is a typical 6,0 decimal field. It is necessary to account for the possibility of a lost leading zero, so a test for the length of the string is done by using the `strlen()` function. If there are only five digits, a leading zero is prefixed. Then, the digits are maneuvered through the `substr()` function, into positions that the `strtotime()` function can process.
3. The third column, “PRODUCTDELETEDATE”, takes a date format that is similar to the preceding date format, only now there is a four-digit year. This is a little simpler because the `substr()` function can predictably run the maneuvers into a format `strtotime()` can process.
4. The fourth column, “PRODUCTLASTUSEDDATE”, takes a DB2 time stamp field and converts it to a SOAP standard time stamp. Now, the `strtotime()` function can process it.

12.4.1 Installing open source applications

Installing an open source application natively on IBM i is simple. There are many industry-standard PHP applications that can be installed on IBM i, including content management systems, wikis, forums, customer relationship management, and much more.

The Young i Professionals open source website contains many resources for helping you learn how to quickly and easily see what applications are available for IBM i, but also provides instructions about how to install these PHP applications. The main website is at the following location:

<http://www.youngiprofessionals.com>

There are instructions about how to install applications such as the following ones:

- ▶ PMWiki: Create and host your own wikis.
- ▶ Pivot: Understand your website traffic with this open source counting tool.
- ▶ WordPress: Open source software to build websites or blogs.
- ▶ Joomla: Industry-recognized Content Management System.

Instructions about how to install these programs on IBM i can be found at the following website:

<http://www.youngiprofessionals.com/wiki/index.php/PHP/PHP>

One of the most successful PHP applications that is running on IBM i today is SugarCRM. It is one of the industry leaders for Customer Relationship Management software.

The SugarCRM application can be used as a standard approach for installing nearly any open source application. Many of the utilities that are needed for the installation are on your IBM i. We explore a handful of those utilities in this example.

Follow the links on the following website to download the sugarCRM software:

<http://www.sugarCRM.org>

You can optionally sign up for more information or follow the links to download the file anonymously. When you have the compressed file, use FTP to send it to the document root of the IBM i. You can expand the extracted directory on your PC and then use FTP, but that takes some time because the SugarCRM is an application with many files.

From the command line, navigate to the document root of the PHP web server, typically /www/zendsvr/htdocs, and run the **jar** command on the file. This **jar** command extracts the files inside the compressed file to a directory that is defined inside the compressed file. The command might take a few minutes, but, when it completes, you can see all the source code that makes up the SugarCRM application.

Follow the documentation on the Sugar website for “On-site installation...”. Verify that the prerequisites are complete and start the installation process. To see the complete instructions for installing SugarCRM on IBM i, see the SugarCRM website:

http://support.sugarcrm.com/04_Find_Answers/02KB/02Administration/100Install/Sugar_on_IBM_i_Installation_-_Configuration_and_Tuning_Guide/

The database prerequisite for SugarCRM on IBM i is MySQL. This database can optionally use the DB2 storage engine to store contents directly into DB2 instead of in MySQL. Although the application thinks it is talking to MySQL, it is storing data directly in the DB2 database and not making a copy.

There are two IBM Redbooks publications that describe the MySQL database: *Discovering MySQL on IBM i5/OS*, SG24-7398 and *Using IBM DB2 for i as a Storage Engine of MySQL*, SG24-7705. They provide many tips about how to navigate the database on IBM i. The current distribution of MySQL for IBM i is called Zend DBi and is available for no charge from <http://www.zend.com> or as part of the Zend Server installation package. Using an open source application such as phpMyAdmin helps in administering MySQL databases.

For a detailed, step-by-step approach, including pictures, see the installation instructions for installing SugarCRM Professional edition, found at:

http://support.sugarcrm.com/02_Documentation/01_Sugar_Editions/04_Sugar_Profession al/Sugar_Professional_7.1/Installation_and_Upgrade_Guide/

Installing a PHP application on IBM i is straightforward and simple. Figure 12-21 shows the SugarCRM interface after you have signed in and SugarCRM is running.

Name	City	Billing Country	Phone	User	Email Address	Date Created
Kaos Trading Ltd	St. Petersburg	USA	(684) 017-0502	Max Jensen	kid.hr@example.it	02/10/2014 06:47pm
Avery Software Co	St. Petersburg	USA	(851) 550-6504	Will Westin	saction.kid@example.edu	02/10/2014 06:47pm
Constrata Trust LLC	Kansas City	USA	(586) 456-4536	Sally Bronson	saction11@example.name	02/10/2014 06:47pm
Draft Diversified Energy Inc	St. Petersburg	USA	(287) 135-9042	Sally Bronson	phona.info@example.biz	02/10/2014 06:47pm
Spindle Broadcast Corp.	Santa Fe	USA	(894) 545-8500	Will Westin	qa76@example.tv	02/10/2014 06:47pm
Q.R.&E. Corp	Persistence	USA	(260) 868-5806	Sarah Smith	kid.ga.info@example.name	02/10/2014 06:47pm
Chandler Logistics Inc	Persistence	USA	(115) 593-9648	Max Jensen	qa37@example.de	02/10/2014 06:47pm

Figure 12-21 Search accounts window

Now that you have the PHP application that is running, you can configure settings for RPG applications to easily access the accounts window. This can be achieved with a simple SQL statement inside MySQL that can be administered through phpMyAdmin.

In phpMyAdmin (Figure 12-22), navigating to the sugarrb database is simple (a list of tables appears on the left). Selecting the **SQL** tab opens an interface to the MySQL Monitor (interpreter). Run **alter table accounts engine=IBMSB2i**, select the accounts table, and click the **Operations** tab to open a UI that can allow many changes. Click the down arrow on the Storage Engine to select IBMDB2I and move all the data about accounts to DB2.

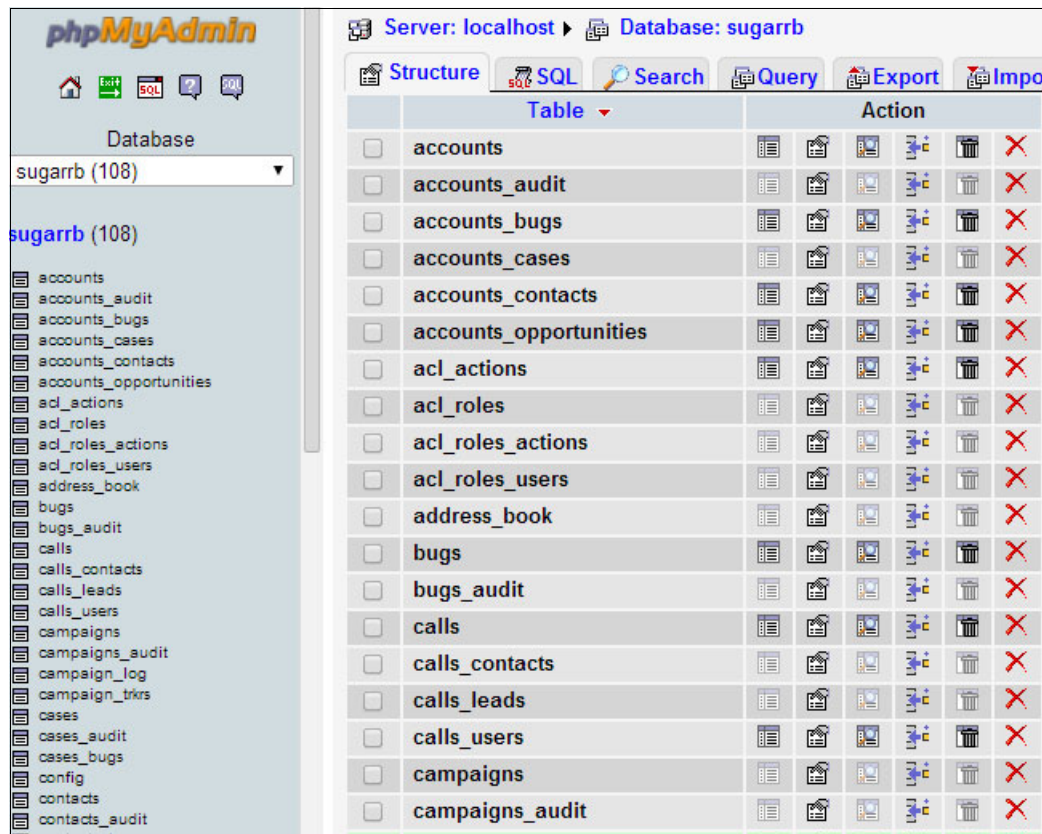


Figure 12-22 PHP Admin window

From the native 5250 screen (Figure 12-23), the objects in the sugarrb library were created automatically. Notice that the indexes that were a part of the MySQL infrastructure also are propagated to DB2.

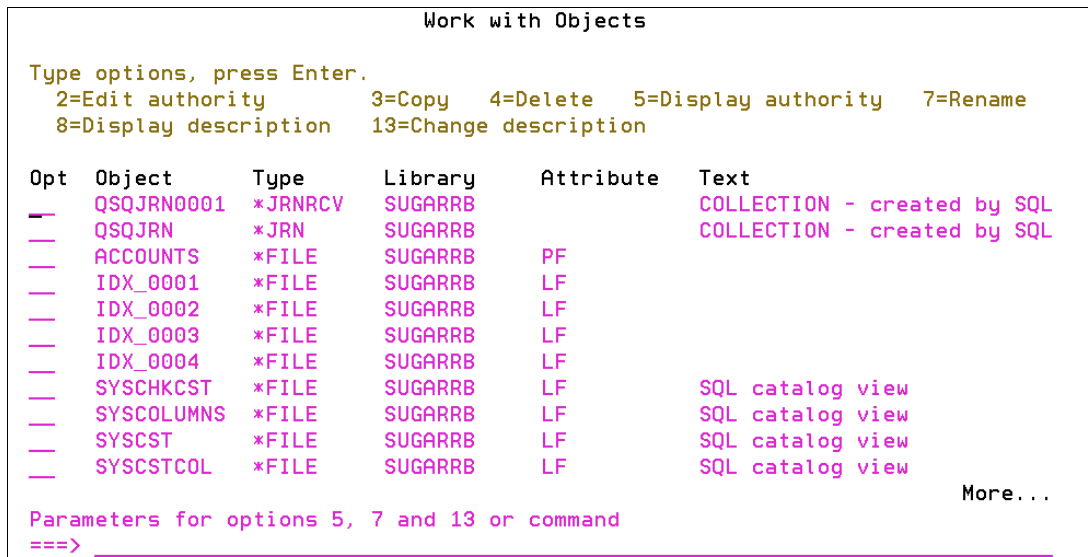


Figure 12-23 Object that is created in the sugarrb library

The account data can be seen in the native environment by running a simple SQL statement (Figure 12-24).

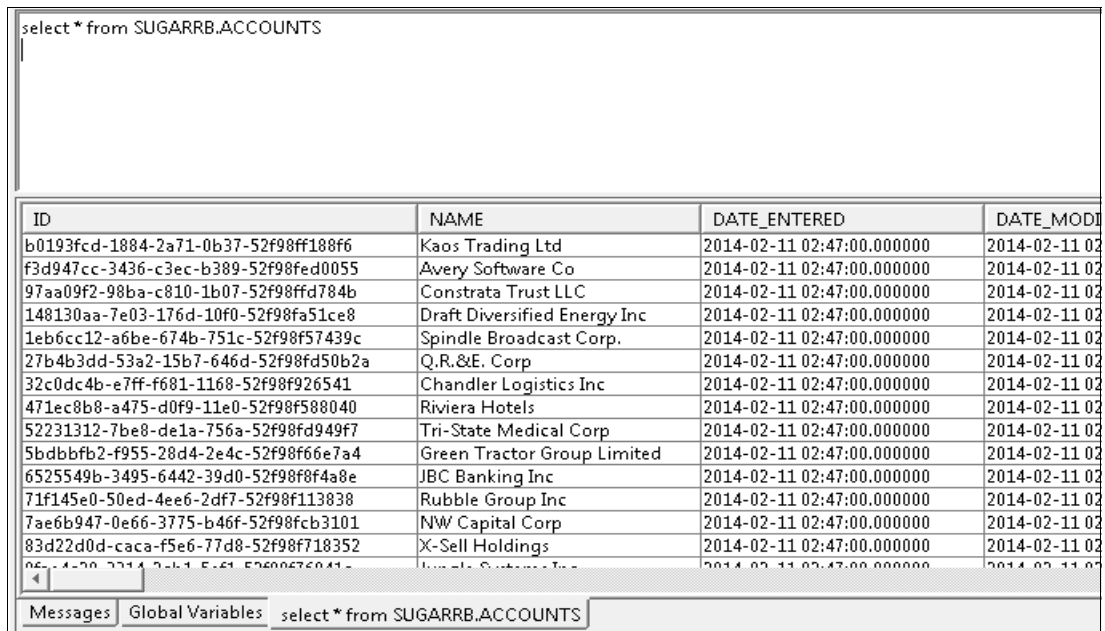


Figure 12-24 Simple SQL statement showing DB content

This type of infrastructure allows RPG and COBOL programs direct access to the tables in the CRM system to support business cases such as automating the creation of new customers, orders, and more.



Java

This chapter provides a brief introduction to Java as an option for application modernization. In this chapter, you find a brief definition of Java and some related concepts. This chapter also provides some reasons to consider Java as the language and platform to modernize your existing applications on IBM i.

13.1 The world of Java

This section introduces the basic concepts of Java and related technologies. These concepts help you achieve a general perspective of Java as a whole and help you to evaluate it as a choice for your modernization needs.

13.1.1 What is Java

The word “Java” can mean different things depending on the context. Java is a programming language *and* a complete development platform that includes an application programming interface (API) and a virtual machine. This section explains what those components are and how they can become useful tools for you in the path to modernization.

Java language

The Java language has been around for a long time and is one of the most popular programming languages in the industry. Java was first introduced in the early 1990s by Sun Microsystems. The main characteristics of the Java language include the following ones:

- ▶ Java is a high-level programming language.
- ▶ Java is an object-oriented language.

The object-oriented paradigm is based on the idea that programs can be designed as a collection of objects interacting with each other. Each object is an entity with a specific and well-defined task to achieve. These objects communicate through methods that have well-defined interfaces.

The object-oriented paradigm can help developers achieve better maintainability, reusability of code, and flexibility. For many developers, this paradigm is the more natural way to design and build applications, especially large-scale solutions.

- ▶ Java is platform-independent.

Java programs are both compiled and interpreted. Compilation translates Java code into an intermediate language called *Java bytecode*. Bytecode is in turn parsed and run (interpreted) by the Java virtual machine (JVM), a translator between the language and the underlying operating system and hardware. A Java program that is written applying preferred practices and good principles can be ported to different platforms without any change. This is especially helpful now that people need to run the same application on different devices (for example, a client-based application).

Java platform

The Java platform is a software-only platform that can run on top of most hardware platforms. It consists of the JVM and the Java API, which is a large collection of ready-made components (classes) that ease application development and deployment. The Java API spans everything from basic objects to networking and security, XML generation, and web services. It is grouped into libraries, which are known as packages, of related classes and interfaces. Figure 13-1 on page 587 shows the architecture of the Java platform.

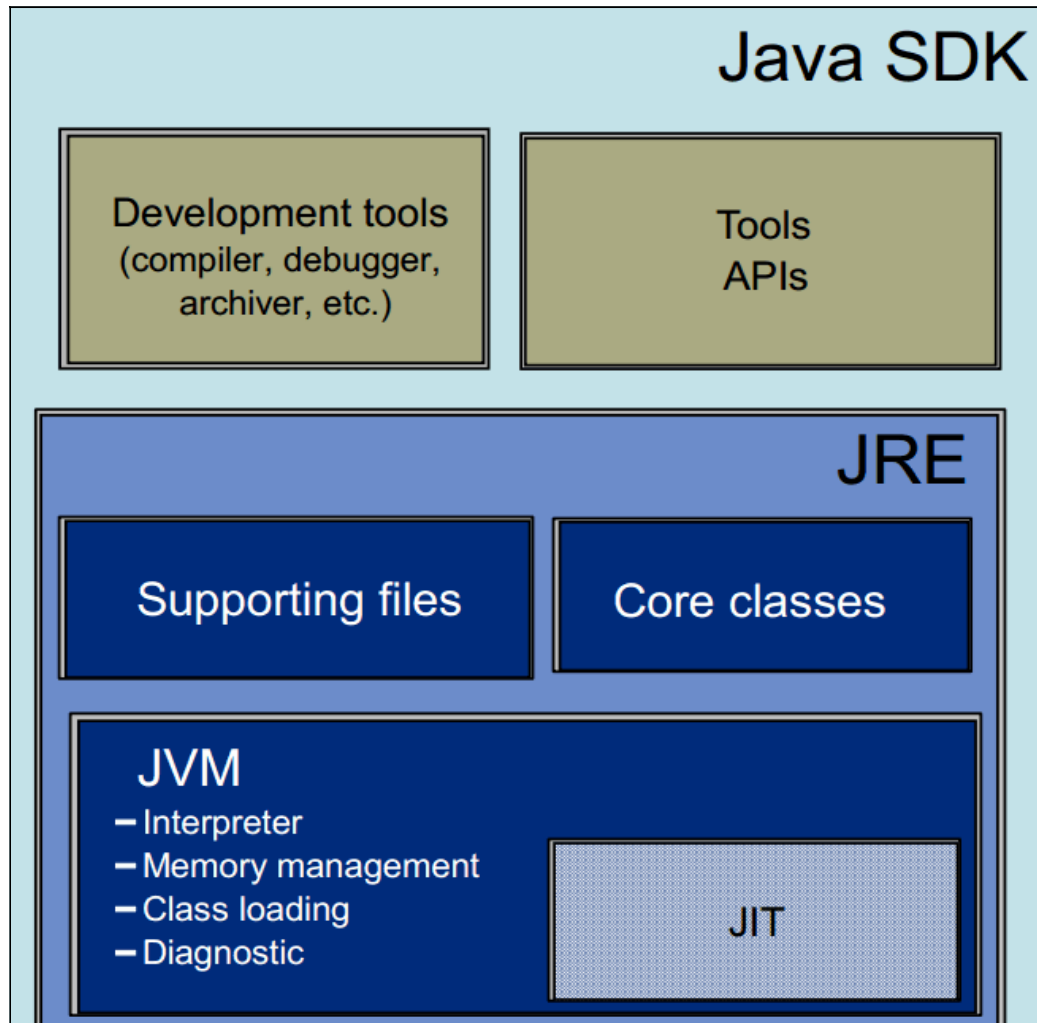


Figure 13-1 Java platform architecture

There are different editions of this platform that you can choose according to the needs of your application:

- ▶ Java Platform, Standard Edition for desktop and server applications
- ▶ Java Platform, Enterprise Edition for web and enterprise applications
- ▶ Java Platform, Micro Edition for applications running on resource-constrained devices

Along with the Java API, every full implementation of the Java platform includes the following items:

- ▶ Development tools for compiling, running, monitoring, debugging, and documenting applications
- ▶ Standard mechanisms for deploying applications to users
- ▶ Toolkits for creating sophisticated GUIs
- ▶ Integration libraries that let programs access databases and manipulate remote objects

For more information about the components of the Java platform, see *IBM Technology for Java Virtual Machine in IBM i5/OS*, SG24-7353.

13.1.2 Java on IBM i

The Java platform has been available on IBM i platform since 1998. Before there was direct Java support in IBM i5/OS V4R2, IBM provided the IBM Toolbox for Java to help Java developers access their IBM i native content from applications running on Java on any platform.

Java applications can run directly on IBM i. Application developers can take advantage of the “write once run anywhere” capabilities that Java provides and make them available on multiple platforms, including the IBM i platform. Since i5/OS V4R2, IBM i has supported Java natively. IBM delivers the latest versions of Java as they become available in the community. Java on IBM i has been supported as part of two different technologies:

- ▶ Classic Java
- ▶ IBM Technology for Java

We cover both of these in depth.

IBM Toolbox for Java and JTOpen

The IBM Toolbox for Java is a library of Java classes supporting the client/server and internet programming model of an IBM i system. The classes can be used by Java applets, servlets, and applications to easily access IBM i data and resources. The toolbox does not require additional client support beyond what is provided by the Java virtual machine and JDK.

The IBM Toolbox for Java provides support that is similar to functions that are available when using the IBM i Access APIs. It uses the IBM i host servers that are part of the base IBM i operating system to access data and resources on an IBM i system. Each of these host servers runs in a separate job on the system, communicating with a Java client program using designed data streams on a socket connection. The socket interfaces are hidden from the Java programmer by the IBM Toolbox for Java classes. Because the socket connections are managed by toolbox classes, IBM i Access is not needed on the workstation.

JavaBeans are provided for most public interfaces. Here is a list of some of the things for which the Java toolbox provides native methods for accessing:

- ▶ Database:
 - JDBC: DB2 for i data can be accessed by using a JDBC driver that is written to the interface that is defined by the JDBC 3.0 specification.
 - Record-Level database access: Physical and logical files on the system can be accessed a record at a time by using the interface of these classes. Files and members can be created, read, deleted, and updated.
- ▶ Integrated File System: The file system classes allow access to files in the Integrated File System. Through the Integrated File System file classes, a Java program can open an input or output stream, open a file for random access, list the contents of a directory, and do other common file system tasks.
- ▶ Programs. Any IBM i program can be called. Parameters can be passed to the IBM i program and data can be returned to the Java program when the server program exits.

A program call framework is provided through a program call markup language (PCML), a tag language that is used for supporting the program call function of the toolbox. The language fully describes all parameters, structures, and field relationships that are necessary to call an IBM i program.
- ▶ Commands. Any non-interactive IBM i command can be run. A list of IBM i messages that are generated by the command is available when the command completes.

- ▶ **Data Queues:** Access to both keyed and sequential data queues is provided. Entries can be added to and removed from a data queue, and data queues can be created or deleted on the system.
- ▶ **Print:** Access to IBM i print resources is provided. Using the print classes, you can retrieve lists of spooled files, output queues, printers, and other print resources. You can work with output queues and spooled files, answer messages for spooled files, and do other print related tasks. Additionally, classes are provided to create spooled files on the system, and to generate SCS printer data streams. Writing directly to these classes, applications, and applets can generate output on the IBM i spool system.
- ▶ **Jobs:** Access IBM i jobs and job logs. Using the job classes, you can retrieve messages in a job log, information about a job (name, number, type, user, status, job queue, and more), or get a list of jobs based on your selection.
- ▶ **Messages:** Access IBM i messages, message queues, and message files. Using the message classes, you can retrieve a message that is generated from a previous operation, such as a command call, information about a message on a message queue, and interact with a message queue allowing you to send, receive, and even reply to messages.
- ▶ **Users and Groups:** Access users and groups. The user and group classes allow you to get a list of users and groups on the system and information about each user.
- ▶ **User Spaces:** Access user spaces. Use the user space class to read, write, create, and delete user spaces on the system.
- ▶ **Data Areas:** Access various kinds of data areas (character, decimal, local, and logical). Entries can be added to and removed from a data queue, and data queues can be created or deleted on the system. Use the data area classes to read, write, create, and delete data areas on the system.
- ▶ **System Values:** Query and reset system values and network attributes on the system.
- ▶ **System Status:** Retrieve system status information. With the SystemStatus class, you can retrieve information, such as the total number of user jobs and system jobs that are currently running on the system, and the storage capacity of the system's auxiliary storage pool.

IBM also released the IBM Toolbox for Java to the open source community as JTOpen. This release enables Java application developers to create applications that access IBM i resources, even if they do not have an IBM i platform in their IT infrastructure. IBM continues to enhance both the licensed program version and the JTOpen versions of IBM Toolbox for Java. The toolbox is a significant help for the Java developer.

Quick tip - IBM Toolbox for Java and JTOpen:

- ▶ IBM Toolbox for Java website:
<http://www.ibm.com/systems/power/software/i/toolbox/index.html>
- ▶ SourceForge download site for JTOpen:
<http://sourceforge.net/projects/jt400/>

The Classic JVM

The Classic JVM was the first implementation of the Java Platform, Standard Edition platform for the IBM i. At that time, developers were tasked with designing and developing a Java platform for the IBM i that would run Java bytecode classes according to the Sun specification, and take advantage of the ease of use, scalability, reliability, and security that IBM i clients have enjoyed for many years. Another key design requirement was optimizing performance. At the time, Java applications were interpreted and therefore had a tendency to consume more processor and memory resources than compiled code.

The IBM i platform uses a technology independent machine interface (TIMI) to separate the operating system and applications from the underlying hardware implementation. IBM developed a set of new instructions below the machine interface specifically to support Java. The actual Java code and API were implemented above the machine interface, as shown in Figure 13-2.

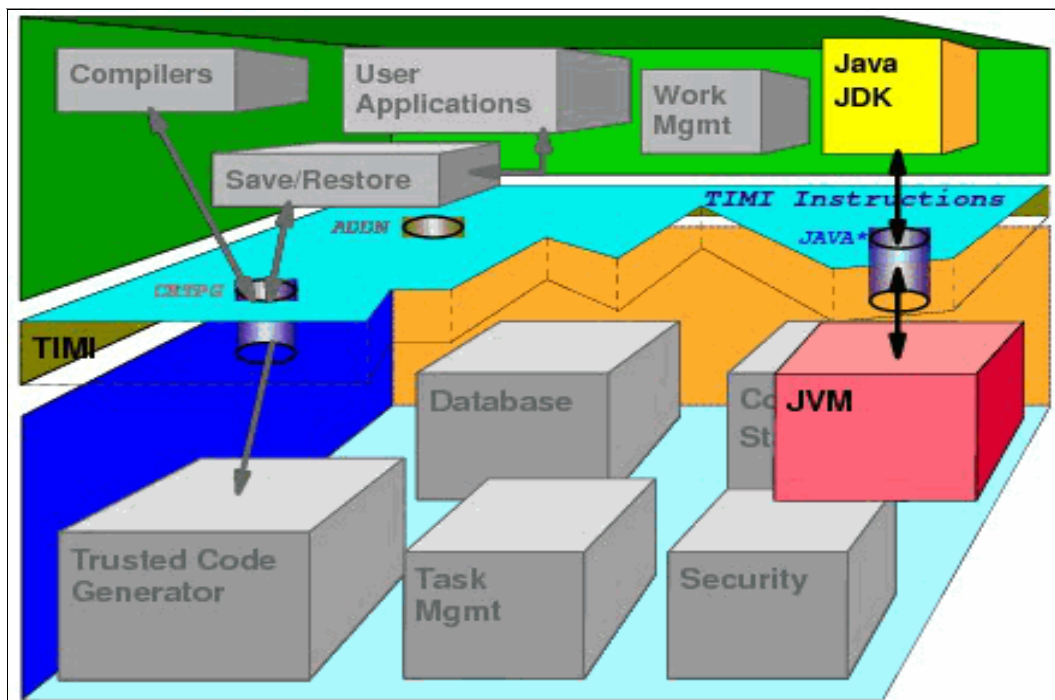


Figure 13-2 Classic Java implementation on IBM i platform

Java on IBM i can take advantage of the 64-bit architecture of the systems to provide a scalable solution from single processor machines all the way up to multiprocessor machines. The Classic JVM was unique in its implementation of asynchronous garbage collection, which allowed the JVM to continue processing application requests during the garbage collection cycle.

Nevertheless, this uniqueness of the Classic JVM became a disadvantage. For many developers and vendors, it required much work to port their applications to the IBM i, which disabled the portability features of the Java platform. This also resulted in more expenses to IBM and slower releases of the new Java versions and updates. In addition, the Classic JVM was based on porting code that was not tuned to the features built within the IBM PowerPC® architecture. Starting in V5R4, IBM i started to move away from the Classic version of Java and support for Classic is now stabilized. Today, Java is delivered on IBM i only through the IBM Technology for Java version.

IBM Technology for Java

Starting with IBM i 5.4, IBM introduced IBM Technology for Java. This new implementation of the Java platform was designed to overcome some of the obstacles that the Classic JVM had. IBM Technology for Java is a highly configurable core JVM that you can use for micro-edition implementations, where a small footprint is of utmost importance, and you can also use it for server and enterprise implementations, where scalability and extensibility are necessary. This JVM has configurable garbage collection policies and advanced performance profiling and optimization features. IBM Technology for Java was created by IBM and is designed to leverage the strength and architecture of the PowerPC processor chip.

Unlike the Classic JVM, IBM Technology for Java is implemented on the IBM i portable application solution environment (PASE), as shown in Figure 13-3.

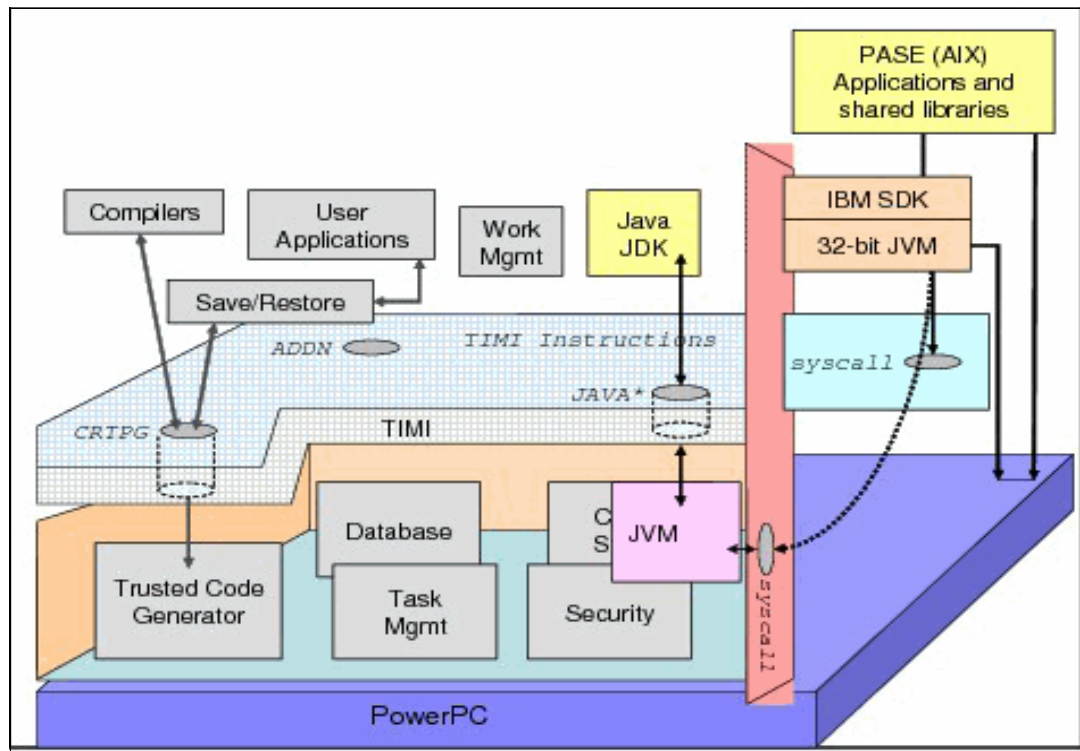


Figure 13-3 IBM Technology for JVM 32-bit implementation on the IBM i

The IBM Technology for Java JVM is displayed as an application in PASE. This environment yields good performance because it is so close to the processor and has a relatively short code path. Because it is not implemented above the IBM i TIMI, it does not have implementation independence from the underlying hardware changes. However, this impacts only the JVM and not user applications.

Initially the IBM Technology for JVM was implemented only on 32-bit architecture, but since IBM i 6.1, the 64-bit version is available for use. This gives more flexibility for developers, allowing them to fit the JVM according to their needs.

IBM Technology for Java is distributed as part of the operating system on the IBM i. On other platforms, Java is an additional component that must be installed separately. This means that JVM on IBM i is shared between all applications, enabling an optimized centralized Java configuration for all of your environment.

Similarities between the JVM implementations

The most important similarity is that both VMs are fully compliant with the JSE specifications, and Java programs can run in either VM without modification or recomputation. You can use both environments within the IBM i Qshell environment to compile (the `javac` command) and run (the `java` command) Java programs.

Both can use the `RUNJAVA` CL command, although some parameters for IBM Technology for Java are ignored. Both can use environment variables that are defined by the IBM i `ADDENVVAR` CL command. Both can use the IBM Toolbox for Java to access resources such as data queues and data areas. Both use the IBM i native Java Database Connectivity (JDBC) driver to access database resources. Both use IBM i debugging APIs that are started through `STRSRVJOB` or `STRDBG` CL commands.

Differences between the JVM implementations

Besides the environment on which they run, there are differences between both implementations.

There are many significant updates to the Java language, including new “state-of-the-art” garbage collection (GC) and Just in Time (JIT) technologies. The new algorithms for these functions are included in IBM Technology for Java. These enhancements have led to a significant improvement in the overall performance compared to the Classic JVM. Additional support for the Classic JVM is stabilized as new features and functions are added only to the IBM Technology for Java edition.

Choosing the JVM implementation

Although the Classic JVM is still available, it has been stabilized. IBM i 6.1 is the last version of the operating system that contains the Classic JVM. Run your Java applications on the IBM Technology for Java version.

Because the Classic JVM is no longer included in newer versions of IBM i, new development and enhancements to the JVMs are being included only in the new IBM Technology for Java version. It is important that you convert your Java applications to the IBM technology for Java edition to ensure that your applications remain a high-performing and secure solution. Additionally, any application running on the Classic JVM does not work when you migrate to IBM i 7.1 or higher.

Starting in IBM i 5.4, you have the opportunity to take advantage of all the IBM Technology for Java version has to offer. In this initial offering, only the 32-bit implementation was supported on IBM i 5.4. At first, this can seem like a significant problem. The Classic JVM version was a 64-bit version (nearly unlimited heap size) and the new version runs with half the address space, meaning a much smaller heap size. With this smaller address space, the 32-bit JVM is limited to about 4 GB of heap space with which to work. The good news is because the JVM is using much less space, the JVM itself is requiring much less heap space. Almost all Java applications can fit in this smaller JVM and can reap the benefits of improved performance and a smaller memory footprint. There are exceptions, but those are the largest applications that are also dealing with large objects.

When you have moved to IBM i 6.1, there are no limitations. Here, both the 32-bit and 64-bit versions of Java are supported, which means that your applications can take advantage of the significant improvements in performance that the IBM Technology for Java versions have to offer. On average, running with this new JVM, you can expect to see about a 30% improvement in performance. When using the 32-bit version, you can expect to see the memory footprint of the JVM to be reduced by 40%. One of the other advancements that was added in IBM i 6.1 is Compressed References (Java 6), which brings many of the performance benefits of the 32-bit JVM to the 64-bit JVM and means that the JVM is using only the extended address space as needed instead of all of the time. New advances like this are another advantage to the IBM Technology for Java version. New levels of Java, for example, Java 7, are delivered only as part of IBM Technology for Java.

Moving to the new IBM Technology for Java version is a simple process. For most applications, it is as simple as changing the Java home for your Java application or for WebSphere Application Servers, a simple matter of running the “enablejvm” script. You want to run a regression test to ensure that all functions are behaving as expected. If you are considering updating your OS level or are planning on updating your applications, it might be time to consider taking advantage of all the improvements waiting for you with IBM Technology for Java.

Supported JVM versions

IBM Technology for Java supports different versions of JDK, depending on the operating system and PTFs that you have. To see the latest details about support for Java on IBM i, see the Java for IBM i topic on IBM developerWorks at the following website:

<http://www.ibm.com/developerworks/ibmi/techupdates/java>

In the following tables, you find a summary of the current available options for IBM i5/OS V5R4, IBM i 6.1 and IBM i 7.1:

► **IBM i5/OS V5R4**

For releases before i5/OS V5R4, IBM Technology for Java is the product 5722JV1 and requires PTF group SF99291. i5/OS V5R4 is no longer covered under normal support. Support is available only through an extended service contract. Options that are displayed in Table 13-1 show what is available.

Table 13-1 IBM Technology for Java options for IBM V5R4

Options	JDK version	JAVA_HOME
8	IBM Technology for Java 5.0 32-bit	/QOpenSys/QIBM/ProdData/JavaVM/jdk50/32bit
11	IBM Technology for Java 6.0 32-bit	/QOpenSys/QIBM/ProdData/JavaVM/jdk60/32bit

► **IBM i 6.1**

For IBM i 6.1, you must have the product 5761JV1 installed with PTF Group SF99562. The options that are displayed in Table 13-2 are available.

Table 13-2 IBM Technology for Java options for IBM i 6.1

Options	JDK Version	JAVA_HOME
8	IBM Technology for Java 5.0 32-bit	/QOpenSys/QIBM/ProdData/JavaVM/jdk50/32bit
9	IBM Technology for Java 5.0 64-bit	/QOpenSys/QIBM/ProdData/JavaVM/jdk50/64bit

Options	JDK Version	JAVA_HOME
11	IBM Technology for Java 6.0 32-bit IBM Technology for Java 6 2.6 32-bit	/QOpenSys/QIBM/ProdData/JavaVM/jdk60/32bit /QOpenSys/QIBM/ProdData/JavaVM/jdk626/32bit
12	IBM Technology for Java 6.0 64-bit IBM Technology for Java 6 2.6 64-bit	/QOpenSys/QIBM/ProdData/JavaVM/jdk60/64bit\ /QOpenSys/QIBM/ProdData/JavaVM/jdk626/64bit
13	IBM Technology for Java 142 64-bit	/QOpenSys/QIBM/ProdData/JavaVM/jdk14/64bit

► IBM i 7.1

If you are running IBM i 7.1, you need the product 5761JV1 with PTF Group SF99572 to enable the options that are displayed in Table 13-3.

Table 13-3 IBM Technology for Java options for IBM i 7.1

Options	JDK Version	JAVA_HOME
8	IBM Technology for Java 5.0 32-bit	QOpenSys/QIBM/ProdData/JavaVM/jdk50/32bit
9	IBM Technology for Java 5.0 64-bit	/QOpenSys/QIBM/ProdData/JavaVM/jdk50/64bit
11	IBM Technology for Java 6.0 32-bit IBM Technology for Java 6 2.6 32-bit	QOpenSys/QIBM/ProdData/JavaVM/jdk60/32bit /QOpenSys/QIBM/ProdData/JavaVM/jdk626/32bit
12	IBM Technology for Java 6.0 64-bit IBM Technology for Java 6 2.6 64-bit	/QOpenSys/QIBM/ProdData/JavaVM/jdk60/64bit /QOpenSys/QIBM/ProdData/JavaVM/jdk626/64bit
13	IBM Technology for Java 142 64-bit	/QOpenSys/QIBM/ProdData/JavaVM/jdk14/64bit
14	IBM Technology for Java 7.0 32-bit IBM Technology for Java 7.1 32-bit	/QOpenSys/QIBM/ProdData/JavaVM/jdk70/32bit /QOpenSys/QIBM/ProdData/JavaVM/jdk71/32bit
15	IBM Technology for Java 7.0 64-bit IBM Technology for Java 7.1 64-bit	/QOpenSys/QIBM/ProdData/JavaVM/jdk70/64bit /QOpenSys/QIBM/ProdData/JavaVM/jdk71/64bit

13.2 Why Java

Why choose Java as your modernization language? This is a valid question, but it does not only apply for the Java language. You can ask the same question for any other programming language. Most of the time, the answer to that question lies in the circumstances and needs of the organization.

Choosing between languages or technologies usually is related to the availability of resources, such as platforms or existing skills. Choose the correct tool for the job. What does it mean? Programming languages usually are classified according to their paradigm. A paradigm is a way of thinking, a perspective about how to solve a problem. Every paradigm has its pros and cons, but depending on the problem that you must solve, some paradigms are more natural than others. For example, the procedural paradigm is used in applications that need higher performance, especially in embedded application or firmware and drivers. If you need to manage complex components and focus on reusability, scalability, and flexibility, maybe you should use an object-oriented language.

IBM i gives you the flexibility to choose between different programming languages that belong to different paradigms, so you have different ways to solve your problems. What you must do is keep an open mind and not think only in terms of one particular language or paradigm. Always search for the best technology for the problem that you need to solve.

Choose Java when it can benefit you more than RPG or any other language on the IBM i. This section introduces some of the main characteristics of Java and its tools that can help you in the modernization journey.

13.2.1 Language and platform characteristics

There are many reasons why you might choose Java to develop your applications. In this section, you learn some of the language and platform characteristics that you should take in to consideration, especially when making decisions in a modernization project.

Object orientation

Java is a language that belongs to the object-oriented programming (OOP) paradigm. In OOP, concepts are represented as objects that have attributes (data fields) and associated procedures that are known as methods. Objects can be viewed as a composition of nouns (like data, such as numbers, strings, or variables) and verbs (like actions, such as functions). Therefore, an object-oriented program is a collection of objects interacting between each other.

The OOP paradigm is based on the following principles that every object-oriented language must implement.

Abstraction

Abstraction is the process by which data and programs are defined with a representation similar in form to its meaning while hiding away the implementation details. For example, when you use a vehicle, you usually are not thinking about every part of it. You are ignoring the complex parts and focusing only on its function as a whole. This is what abstraction is about.

In the context of OOP, you start defining objects abstracting only the parts that you need without having to define huge complex entities. Figure 13-4 shows an example of abstraction. As you can see in this example, the MotorVehicle object does not contain all the details about a real motor vehicle. The abstraction is focusing only on the functions and attributes that are relevant in a specific situation.

MotorVehicle
-fuelLevel
+accelerate() +break() +steerLeft() +steerRight()

Figure 13-4 MotorVehicle object abstraction

Encapsulation

Encapsulation is a mechanism to bind together code and data and to restrict its access. Encapsulation means that the internal implementation of the object is hidden from the outside. Data for an object is accessible only through the object's methods.

In the Java language, encapsulation is achieved through classes. A class defines the structure and behavior of an object. An instance of a class is a particular object that is created from a class. Going back to the car example, you can think of a class as a particular model of a specific brand, and an instance of that class is the car with particular characteristics, such color, style, and fuel consumption.

Inheritance

Inheritance is a mechanism by which an object acquires the properties and functions of another object. Inheritance is a natural way to define relationships between objects. A simple example of this concept is animals. For example, a dog is part of the mammal class, which is part of the animal class. A dog shares many characteristics with other mammals, and all the mammals share many characteristics with all animals.

Inheritance allows you to split the complexity of the objects through hierarchical classifications. Every object acquires the attributes and methods of its parent. So, for example, if you create the Truck class, it is going to have the same methods and attributes as the MotorVehicle class, without duplicating the code. Also, the Truck class can have its own attributes and methods. This concept is called specialization. Figure 13-5 on page 597 shows an example of inheritance.

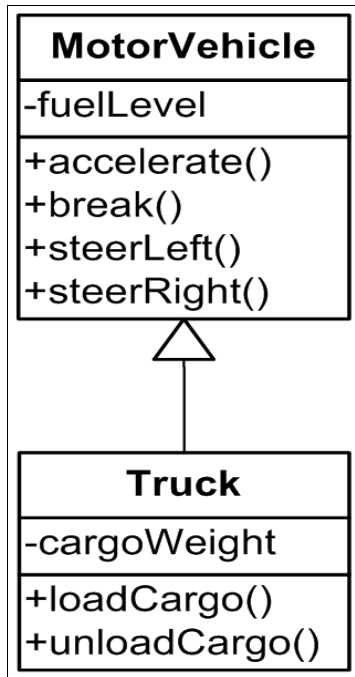


Figure 13-5 Inheritance sample

Polymorphism

The term polymorphism can be defined as the ability of an object to take many forms. In the context of programming, polymorphism ensures that the correct method is called for an object of a specific class when the object is referred to as a more general type. To understand what this means, consider Figure 13-6.

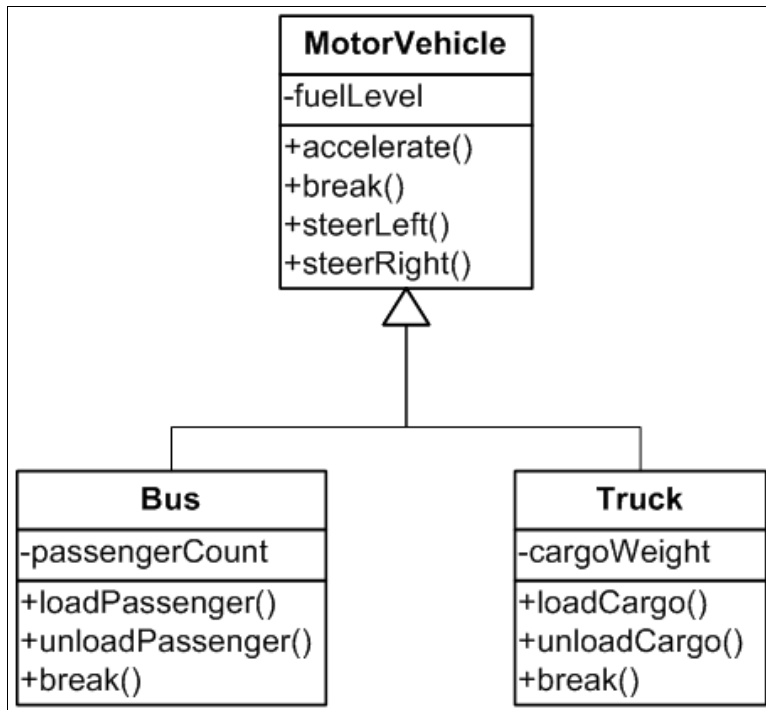


Figure 13-6 Polymorphism example

In this example, the Bus and Truck classes are extending the function of the MotorVehicle class. As you can see, all classes have the break method. With polymorphism, a program can declare an object of class Truck but refer to it as a MotorVehicle. When the program calls the break method, polymorphism makes sure to call the correct break method, in this case, the method that is implemented for the class Truck and not the method of the MotorVehicle class.

Polymorphism can be difficult to understand when you have not used it. However, as soon as you create a class that uses it, you begin to notice the flexibility that OOP can give you.

Platform independence

One of the most important features of the Java language is that the same code can be run on different platforms without changes. To take advantage of this feature, your application must be well-designed by following preferred practices. Otherwise, you must make some changes in your code.

Platform independence can help you port your application to different environments and operating systems, which can provide many benefits, especially if you are developing rich-client applications.

Architecture

In Java, object-oriented principles can help you create good architecture design. When you apply the following principles (abstraction, encapsulation, inheritance, and polymorphism), you can design and build applications that exhibit characteristics such as the following ones:

- ▶ Flexibility

With OOP, you can easily create objects with loose coupling, which means that you can change lower-level objects without having to recompile or build higher-level objects that use them. Java natively supports different ways to implement this flexible behavior.

- ▶ Reusability

In Java, it is easier to design and build reusable components. The grade of reusability depends on the quality of the design choices that you made.

- ▶ Tiers

You can achieve real *n-tier* applications with Java. N-tier architecture is based on the concept of separation and easy replacement of the layers of the application. Java allows this architectural approach in a straightforward way.

Using a non-object-oriented language is no excuse to design monolithic applications without any sense of architecture. Preferred practices of programming go beyond the boundaries of languages and technologies. Java is useful for creating good architectural design that allows scalability, flexibility, and reusability.

13.2.2 Tools

The usage of modern tools is essential in any modernization effort. Tools can help you work better and faster. Many tools are available for the Java language, from development environments to application profilers. There are many tools that are available at not charge, and also many vendors who create many tools for Java.

This section provides you with a quick overview of some of the tools that are available for Java developers. Many of these tools are available for multiple platforms, including the IBM i.

Integrated development environment

The integrated development environment (IDE) is an essential tool for any modern developer. IDEs can help you in many frequent tasks:

- ▶ Compiling and building your application
- ▶ Debugging
- ▶ Refactoring
- ▶ Understanding code
- ▶ Building automatic tests
- ▶ Editing with modern capabilities

For the Java language, there are many IDEs available, ranging from basic products to full enterprise tools. On the IBM i side, IBM Rational Developer for i RPG and COBOL + Modernization Tools, Java Edition V9.0 provide a professional set of tools to design and build RPG, Cobol, and Java on the IBM i.

Quick tip: For more information about IBM Rational Developer for i, see the following website:

<http://www.ibm.com/software/products/en/dev-ibm-i/>

Java IDEs are a rich tool for any developer. They can help you reduce the time that it takes you to do many simple tasks, such as writing code, finding new methods, compiling, and debugging. Java IDEs can be integrated with many plug-ins that can help you create a true integrated development environment.

Profilers

Profilers, in the context of software, are tools that are specialized for doing dynamic program analysis that measures memory, complexity, usage of particular instructions, frequency and duration of function calls, and other things. Profilers are used to help the programmer to optimize software.

Profilers are essential tools for developers who are dealing with performance issues. They generate more than statistical data. For Java, profilers can help optimize the application and tune the configuration of the platform.

IBM Monitoring and Diagnostic Tools for Java

The IBM Support Assistant is a workbench that is based on Eclipse and simplifies software support. It provides access to many different tools that can assist in different areas of problem diagnostic tests. IBM Support Assistant contains the IBM Monitoring and Diagnostic Tools for Java. These tools can help you understand, monitor, and diagnose problem applications running on Java.

IBM Monitoring and Diagnostic Tools is distributed as a no charge suite of tools that includes the following functions:

- ▶ Garbage Collection and Memory Analyzer:

This tool analyzes Java and its heap memory. It creates graphs to show garbage collection and Java heap statistics over time. It is helpful for memory errors and performance tuning.

Here are some highlights of the Garbage Collection and Memory Analyzer:

- Graphical display of data
- Use of heuristics to guide the user toward issues that might be affecting the performance

- ▶ Health Center

Health Center is a live monitoring tool with low processing impact. It can help you understand how your application is behaving and diagnose potential problems. It is suitable for all Java applications.

- ▶ Memory Analyzer

The Memory Analyzer is a tool for analyzing heap dumps and identifying memory leaks from JVM.

- ▶ Interactive Diagnostic Data Explorer

This tool is the strategic tool for allowing interactive analysis of JVM problems using post-mortem artifacts. It is lightweight, allowing you to quickly get information from the artifacts that you are investigating when you are not sure what the problem is and you want to avoid starting a resource-intensive analysis.

13.2.3 Documentation and community

Before you adopt any specific technology, it is a good idea to determine the availability of related documentation and the size of its community. Both elements are great resources that can help you learn, build better software, and find solutions to your problems faster.

The Java language and its platform have been adopted by many developers, which constantly generates much documentation. Here are some of the available resources:

- ▶ Official documentation (including Oracle and IBM documentation)
- ▶ Books and papers
- ▶ Magazines
- ▶ Blogs and forums
- ▶ Tutorials
- ▶ Libraries
- ▶ Open source tools

If you choose Java as your modernization language, you have many resources at your disposal. This is one of the strongest characteristics of Java.

13.2.4 Freedom of choice

When considering a specific development technology, you should know how many resources you have at your disposal. In this chapter, you learned about many of the tools and the documentation that are available for Java, but you should also consider resources such as frameworks, design patterns, and libraries that you can use when designing and building your applications.

At the beginning of this chapter, you learned about how reusability is a part of the Java language. The Java community publishes many libraries, frameworks, and design patterns that you can use at no charge. You can create good software designs yourself, but sometimes using some of these resources can help you increase your productivity.

Java has many resources for you, including web development frameworks, utility APIs, and database access frameworks. Most of them are open source and you can use them if you must.

This section briefly introduces some of the most common resources that are available for the Java language.

Web application frameworks

Web application development involves many repeatable tasks. There are many frameworks for Java that help you develop web applications faster and better. These frameworks provide reusable components for security, sending and receiving emails, session management, HTML generation, and other things. Here are some of the most common frameworks for web development:

- ▶ Struts 2
- ▶ JavaServer Faces (JSF)
- ▶ Spring MVC
- ▶ Google Web Toolkit (GWT)
- ▶ Grails

Persistence frameworks

Most of the applications use a relational database to store their data. Persistence frameworks help the programmer map the database relational representation to object representation of data. Known as *object-relational mapping* frameworks, these frameworks bring the following benefits:

- ▶ Development time reduction
- ▶ Decoupling from the underlying database server

Libraries

A library in Java is a set of reusable classes. The Java API is composed of many libraries that are reusable by the programmer according to the needs of the application. Here are some of the functions of the libraries that are available in the Java API:

- ▶ Advanced user interface components (Java FX)
- ▶ 3D graphics
- ▶ Email sending and receiving
- ▶ Speech recognition
- ▶ Image manipulation

In addition to the core Java libraries, there are many external libraries that are available on open source websites.

13.2.5 Skills and new talent

From an academic point of view, most computer science students learn Java throughout their years of education. It is one of the preferred languages to teach OOP. This does not mean that every new graduate is proficient with Java, but at least this makes it more likely that they have experience working with the language and the platform itself.

Many people who develop software on the IBM i platform use a traditional language such as RPG and do not have a background working in Java or another object-oriented language. If you choose Java as your modernization language, you should have a number of people, from inside and outside of your company, who have used the language before, so creating skills should not be too difficult.

The availability of new talent and skills for the Java language does not mean that this should be the language to choose. This is an advantage of going through the Java path. Many companies have employed new talents with a strong Java background and made the transition to RPG seamlessly.

If you want to start using Java on the IBM i, remember that, in this book, you have been introduced to techniques to modernize applications, mostly written in RPG. If you start applying what you have learned so far, the transition to Java is much easier.

13.3 Moving to Java

You must learn some basic concepts of Java and many of the benefits that it can give you. In this section, you learn some considerations that you must consider when deciding whether to use Java.

13.3.1 The learning curve

“Is Java hard to learn?” That is the question many people ask before starting down the Java path. The truth is that the answer is subjective. Java is a different language and a different programming paradigm, so it is not as simple as learning the grammar and syntax of the language. You must think in an object-oriented manner.

For most new talent, the learning curve is likely to be easier because they probably begin learning programming with the OOP paradigm. However, even if you are more familiar with procedural languages, learning Java and object orientation should not be too difficult. OOP is a more natural approach to problem solving.

Some people find it difficult to learn Java because they are used to coding in a specific way, using obsolete techniques and creating large, monolithic applications. However, if you are using good modular programming techniques, you find many concepts of OOP familiar. For example, consider ILE. With ILE, you had to learn many modularization concepts to create a good design and receive its benefits. If you are used to creating small modules with only one responsibility and to pack only related modules in a service program, you can move from ILE to OOP easily.

What can be harder to learn are all the additional things that you need to create your application, such as tools, web application servers, and database access components. Avoid trying to learn everything at once. Start simple and add complexity step-by-step. For example, in Java there are many frameworks. If you try to learn all of them, you probably end up feeling overwhelmed. Try to focus on the pattern of the framework (the underlying techniques) and you see that most of them are similar.

In summary, the learning curve for Java implies the following aspects:

- ▶ Learning how to think in the OOP paradigm
- ▶ Learning the Java grammar and syntax
- ▶ Becoming familiar with the tools
- ▶ Understanding the frameworks for web and database access

After a while, you become more efficient in the Java world and you receive many benefits. You likely see a reduction in development time and costs. The Java world has many tools that help you automate many tasks and also many resources to help you avoid reinventing things.

This chapter is not stating that Java is always the best approach. You must use the correct language for the needs of the project. Some problems are more natural to resolve in a procedural way, but many of them are more likely to be resolved in an OOP paradigm. Choose wisely.

13.3.2 Myths surrounding Java

There are many myths that scare people away from Java. Some of these myths were true when the language was first invented. However, the Java language and the platforms have been optimized. IBM has invested much time and money to build a JVM for their systems, including the IBM i (IBM Technology for Java is specifically created to run on the PowerPC architecture). In this section, you learn about some of the most common myths surrounding Java and you can reassess whether they are true.

Java is slow

Is Java really slow? It depends on what you compare it to. Early in this chapter, you learned that you must choose the correct tool for the job. If what you need is high performance, you must select a lower-level language, such as C/C++ or RPG, but if you are dealing with huge and complex applications, it might be better to use a language with more flexibility, such as Java.

The reputation for slowness that Java usually carries is related to the JVM and not the language itself. The Java language is dependent on multithreads and large amounts of memory. Many people who are accustomed to running other languages starve their Java applications, which causes terrible performance. On the IBM side, the Classic JVM was not designed for IBM POWER architectures. It was ported from the original Oracle version, which resulted in performance issues. However, IBM Technology for Java is designed for the platform and it has been highly optimized by IBM to leverage the Power architecture. With this new version of Java, you can take advantage of the multithreading nature of Java, which was not possible before. Lastly, processor technology has improved greatly over the past few years. Processors are geared toward multithreading, which is a significant boost to Java based applications.

Java is hard to tune and debug

Many people claim that Java is hard to tune, especially on the IBM i platform. However, the truth is that IBM has made available many performance tuning tools that can help you. These tools can help you diagnose your application and the JVM configuration, so you can start changing your current settings to obtain the best results.

The IBM Monitoring and Diagnostic Tools for Java is a no charge tool that can help you analyze any issue with the JVM configuration or in your application. This is an essential tool for any developer and it eases tuning Java on IBM i running IBM Technology for Java.

IBM Technology for Java is highly customizable and easy to tune. There are multiple garbage collection algorithms that you can choose according to your application. For more information about the garbage collection policies and other performance tuning techniques, see *IBM Technology for Java Virtual Machine in IBM i5/OS, SG24-7353*

Java is legacy

Some people claim that Java is now a legacy language. The reality is that it depends on what they mean by legacy. If you think of legacy as an age measurement, most programming languages are legacy. Java is continuing to advance. Oracle is constantly updating the language with new features and IBM is investing many resources to optimize the JVM to the latest POWER processors and make it compliant with the latest releases of Java language.

Java is not invulnerable to bad design and bad coding. You can create code with Java or any other language that exhibits the same issues as the traditional legacy applications. It is not a matter of language. It is a matter of how you code.

13.3.3 Further reading

So far you have been introduced to some basic topics about the Java world. You also learned about things to consider whether Java is going to be your modernization technology. However, this is only the beginning. You must keep learning. The following resources might be helpful:

- ▶ Java on IBM i:

<https://www.ibm.com/developerworks/ibmi/techupdates/java>

- ▶ IBM technical resources for Java developers, including tutorials, forums, and open source projects:

<http://www.ibm.com/developerworks/java/>



Web serving: What and why

Web serving is an important technology that must be considered when you are modernizing an application. There are many ways to create a web presence on IBM i. This chapter covers some of the alternatives and the benefits of the different technologies at a high level. Whatever method is used, accessing applications from the web is a key aspect to modernization in today's world.

14.1 Web server versus application server

These two terms are often used interchangeably, which leads to a degree of confusion. This section looks at the difference and roles of both the web server and the application server. It is important to understand the role of each in the context of the website.

14.1.1 Web server

A web server is an important component for accessing content on the web. When a user accesses an application from a browser, that URL request is sent out into the internet and routed to a web server. The web server is primarily responsible for receiving that request and processing it. The web server handles many different facets of that request. Consider some of the roles and responsibilities of the web server:

- ▶ **Security:** This is the primary role of the web server. Within the web server container, you can configure things to ensure that a URL request can access only the designated or approved content on your server. This keeps the rest of your system safe from potential hacking.
- ▶ **Encryption:** If you are concerned about data from your application being displayed in the clear across the internet, you can configure the web server to encrypt the data. Basically, the data is scrambled with a special key. The browser on the other end of the request unscrambles the content and displays it in a readable manner.
- ▶ **Load balancing:** For many applications, if the number of requests exceeds the capacity of a single server, the web server can be used as a load balancer to distribute requests to different back-end servers to help ensure a consistent user response.
- ▶ **Content serving:** The web server is the tool for building and displaying content back to a user in a browser. The web server is responsible for displaying static page content. It returns HTML, images, video, pictures, and other content to the user. The web server is fast at finding and returning any type of static content. Often, cache methodologies are used to ensure that high use static content is kept in a place that ensures a fast return to the user. Then, the application content must return content that is not static, that is, dynamic, which is where things such CGI or application servers are used.
- ▶ **Front door:** This is not a technical term, but it is an important role that the web server often plays. Requests come in from the internet. Many times, the web server is kept outside of the firewall so it can catch any request that is directed to it. The web server is then responsible for validating the request to make sure that it is for something that it knows about. It also verifies that the user for this request is authorized. After the web server has done some of this basic verification, the request is processed. Often, this process means that the request is passed through the firewall to the back-end server to complete the request. The back-end server can be a number of different things. For example, you call more traditional programs using a CGI methodology. Alternatively, this is where the application server is used.

In the industry today, multiple web servers are available. The rest of this section describes some of the options.

Apache

The Apache HTTP Server Project is a collaborative software development effort that is aimed at creating a robust, commercial-grade, feature-rich, and freely available source code implementation of an HTTP (web) server. The project is jointly managed by a group of volunteers who are located around the world, using the internet and the web to communicate, plan, and develop the server and its related documentation. This project is part of the Apache Software Foundation. In addition, hundreds of users have contributed ideas, code, and documentation to the project. This is the foundation for the IBM HTTP Server for i.

On IBM i, there is only one option for web serving. The IBM HTTP Server Powered by Apache (5770- DG1) is the web server for IBM i. The IBM HTTP Server uses the open source Apache server. The source code from the Apache foundation is ported to IBM i. Everything is recompiled into a native program object, and the source is updated to use the specific IBM i native support where it makes sense. Many IBM i only features are also incorporated to help ensure that IBM i web users have a secure and reliable container for working with the web on IBM i.

Figure 14-1 shows the common deployment of the different components when using web-based applications. The HTTP server is often the front end to the application, and is often situated outside the firewall to help ensure a safe environment.

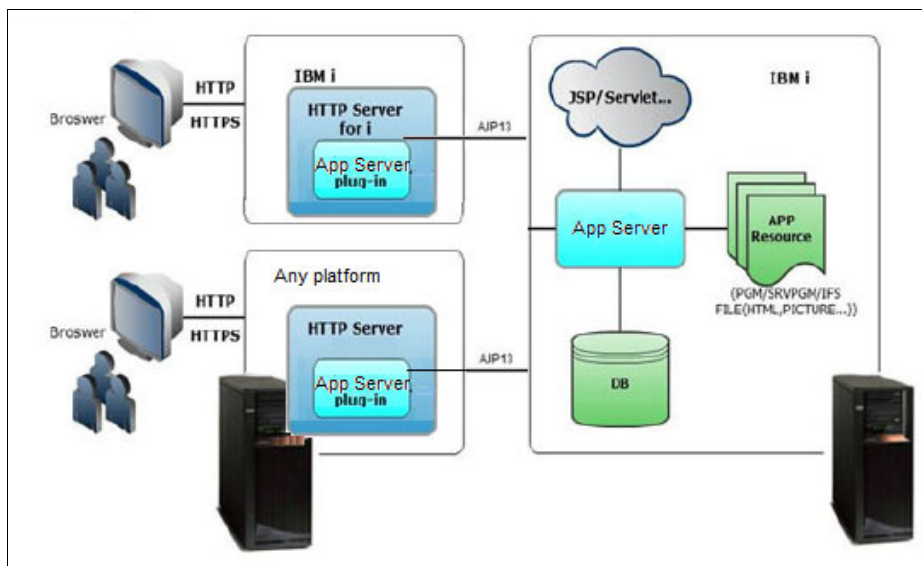


Figure 14-1 Application server configuration

14.1.2 Application server

An application server provides the infrastructure for creating highly dynamic and responsive applications that run your business. The application server in many ways can be thought of as the operating system for the web applications. It insulates the application from the hardware, the operating system, and the network. Like the web server, the application server performs many different roles. In simpler terms, an application server serves as a platform to develop and deploy your dynamic web applications, web services, and EJBs. It is responsible for building the pages that are displayed, gathering the content and data as required, and providing the user interface engine. Application servers can be used as the UI part of the application, using back-end applications or using more direct database access to build the page content dynamically, including the necessary business logic. It can also serve as a transaction and messaging engine. Additionally, the application server can deliver business logic to users on many client devices. Figure 14-2 illustrates the basic presentation of an application server and its environment.

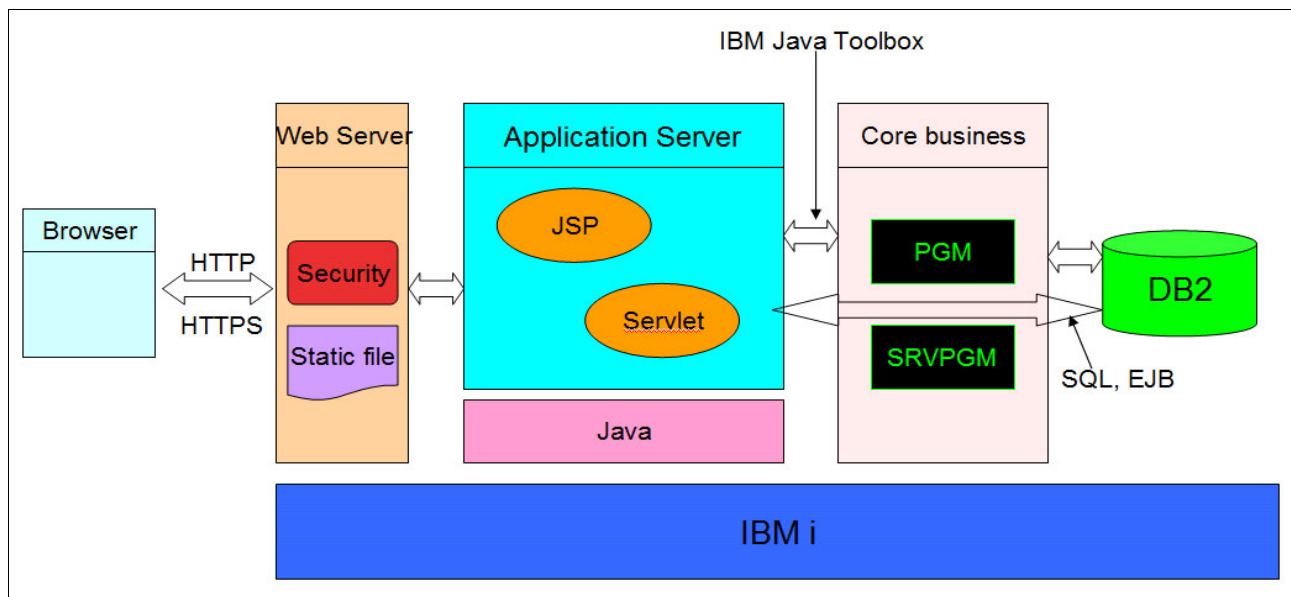


Figure 14-2 Basic presentation of an application server and its environment

The application server acts as middleware between back-end systems and clients. It provides a programming model, an infrastructure framework, and a set of standards for a consistent designed link between them.

There are many different types of application servers. Servers are based on a wide array of languages, including Java, PHP, and Ruby. This section looks at some of these different containers to help you determine what container makes sense in your environment.

14.2 WebSphere Application Server

WebSphere Application Server is the industry leader for enterprise application serving. It is a Java based application server that can provide a solution for the smallest and simplest web applications right up to the most complex, high-performing applications in the industry. The first version of WebSphere Application Server for IBM i was Version 3.5. Since that time, every release of WebSphere Application Server has been delivered on IBM i on the same date as every other supported platform.

WebSphere Application Server is the exact same run time on IBM i that is run on other platforms. All the functions and behavior are the same on IBM i as for other platforms. The only difference on IBM i is the licensing. When you purchase an IBM i operating system license, you are entitled to a license of WebSphere Application Server Express. The only difference on IBM i with Express versus other platforms is that on IBM i there are no processor restrictions. You can start as many instances as you want. If you require additional functions beyond what is offered with the Express version, you must purchase and move to the Network Deployment version of WebSphere Application Server.

When you are ready to use WebSphere Application Server, you must use the Installation Manager for the Rational Software Delivery Platform. On IBM i, the process is a bit more difficult than other platforms because the GUI interface for does not work for IBM i. If you want to use it to install the application server, you must use the command line and response file method.

However, there is another installation method. The IBM i platform has an integrated web administration GUI interface (Web Admin). This interface is reached through the port 2001 interface at the following URL on your system:

http://hostname:2001/HTTPAdmin

Click the **Manager** tab and then the **Installations** subtab, as shown in Figure 14-3.

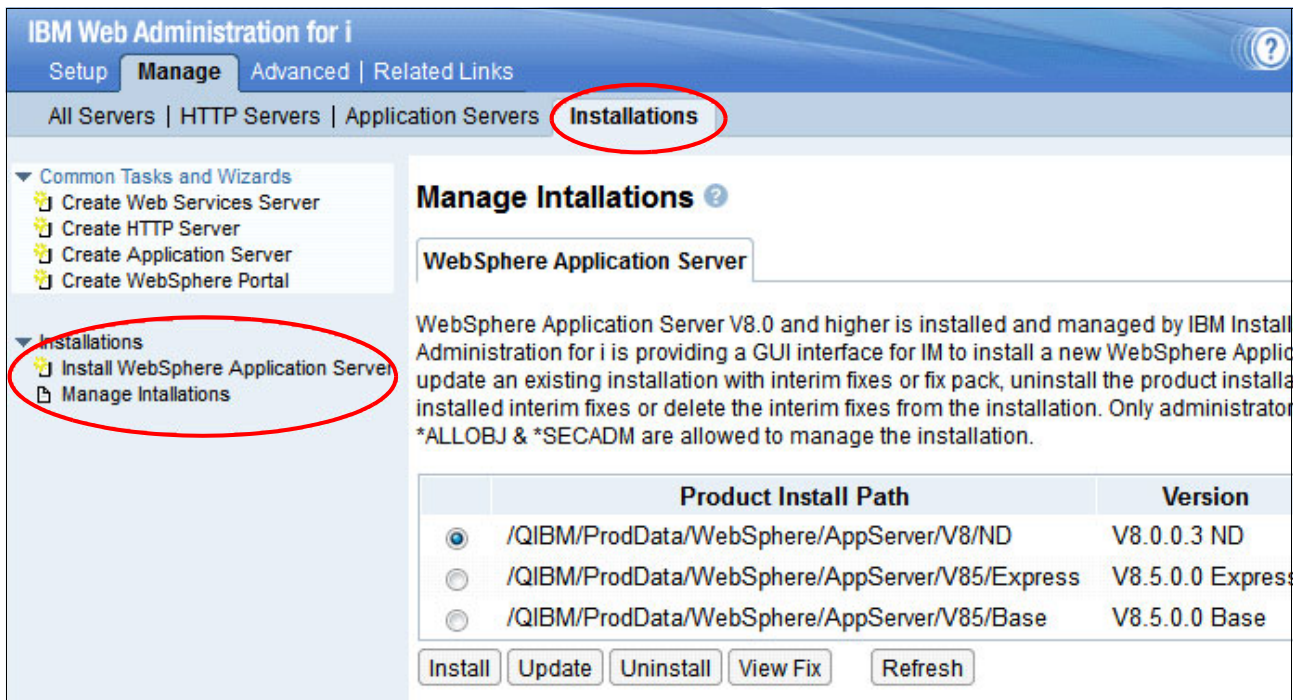


Figure 14-3 IBM i Web Administration web page

To install the wanted version of WebSphere Application Server (because you can have *many* installed and independently running on IBM i), click **Install** to start the installation wizard. This wizard asks you for the location of the installation media and then takes care of the rest of the installation for you. Using the Web Admin GUI installation process greatly simplifies the installation and configuration of your application servers. You can continue to use the Web Admin GUI when it is time to load and install WebSphere Application Server fix packs, to create a server instance to use, and to start and stop all of your WebSphere Application Servers.

The latest version of application servers for the WebSphere family is called *Liberty*. The liberty profile is intended to be a lightweight, deployable, and developer-centric web container. It is perfect for the developer who is working to quickly build and develop applications on any platform. Liberty starts in seconds, and applications can be hot-deployed. Drop the updates in the application directory and they immediately start. Liberty is a great place to run simple production applications or a place to develop applications before they are migrated to the enterprise versions of the WebSphere Application Server family.

WebSphere Application Server provides the Java based environment to run your solutions and to integrate them with every platform and system as business application services that conform to the SOA reference architecture.

WebSphere Application Server Network Deployment provides the capabilities to develop more enhanced server infrastructures. It extends the base package of WebSphere Application Server and includes the following features:

- ▶ Clustering capabilities
- ▶ Edge components
- ▶ Dynamic scalability
- ▶ High availability
- ▶ Advanced centralized management features for distributed configurations

Because different application scenarios require different levels of application server capabilities, WebSphere Application Server is available in multiple packaging options. Although these options share a common foundation, each provides unique benefits to meet the needs of applications and the infrastructure that supports them. At least one WebSphere Application Server product fulfills the requirements of any particular project and its supporting infrastructure. As your business grows, the WebSphere Application Server family provides a migration path to more complex and scalable configurations.

On IBM i, the following packages are available:

- ▶ WebSphere Liberty
- ▶ WebSphere Application Server - Express
- ▶ WebSphere Application Server
- ▶ WebSphere Application Server for Developers
- ▶ WebSphere Application Server Network Deployment

14.3 Integrated Web Application Server

The Integrated Web Application Server (IAS) has been included in the IBM i operating system since Version 5 Release 3. The IAS server is intended to be a minimal footprint, easy to configure, and secure infrastructure for hosting Java web applications on IBM i. IAS enables a subset of the full Java 2, Enterprise Edition specification; if your application uses static web components (HTML, images, documents, and so on), servlets, JavaServer Pages (JSPs), or JavaServer Faces (JSF) components, this integrated alternative might be the best place to run your applications. In addition to providing a runtime container for your own applications, it also provides a place for IBM applications to run without requiring you to deploy, configure, and maintain an additional web container. Applications such as DB2 WebQuery, Web Administration GUI, IBM Navigator, Application Runtime Expert, Integrated Web Services Server, and others can run.

For the past several releases of the IBM i operating systems, the IAS server has been based on a technology called *Lightweight Infrastructure (LWI)*. This is an application server that is based on *Open Services Gateway initiative (OSGi)* technology. OSGi allows you to have a portable lightweight container. It is based on a building block of technology called *bundles*. A bundle is a common application component that can be used by the entire application container. The IAS server allows you to build only the application technology that is required for this application, keeping things streamlined. You must start and use only the technology that is required by the applications. Many servers give you everything, regardless of the needs of the application.

Starting in IBM i 7.1, the IAS server is migrating from an LWI base to the new WebSphere Liberty base. This is an important move because IBM i continues to provide a state of the art web container. The IAS server is intended to be the place where you run or develop your simple web applications. Using the Liberty technology ensures that your applications can migrate to a full WebSphere Application Server container much easier. The upgrade path from Liberty to WebSphere Application Server can move your applications. Additionally, It is now much easier to develop your Java applications on any platform using Liberty because it is portable to many platforms. You can develop your application on your workstations and seamlessly move it to your enterprise hardware.

To access and manage your integrated web application servers, you need to use the Web Administration GUI interface (Web Admin), found at the following URL on your system:

`http://hostname:2001/HTTPAdmin`

From this web-based management console, you can create, start, stop, install, and use other basic management functions for the integrated application server.

14.4 Tomcat

Apache Software Foundation Tomcat (Tomcat) is an open source web server and servlet container that is developed by the Apache Software Foundation (ASF). Tomcat runs with great performance and is a collaboration of the best-of-breed developers from around the world. Compared to the previous version, Tomcat V7 implements the Servlet 3.0 and JavaServer Pages (JSP) 2.2 specifications. It offers many enhancements for most operating systems, including IBM i. A number of years ago, IBM i included Tomcat on IBM i. The last version that IBM shipped was Version 3.2.4. One of the major reasons for not including Tomcat in IBM i is that Tomcat is a pure Java application and can be quickly and easily downloaded and started on IBM i with little to no effort. In the IBM i zone on the developerWorks website, there is a great article that walks through installing and running Tomcat on IBM i. You can find that article at the following website:

<http://www.ibm.com/developerworks/ibmi/library/i-websolution-asf-tomcat/index.html>

There are thousands of IBM i users who are running Tomcat on IBM i today.

14.5 PHP: Zend Technologies

Some years ago, the IBM i team formed a partnership with Zend Technologies to deliver the most popular web infrastructure in the industry on IBM i. This was an important partnership for the IBM i family. First, having PHP run in a native manner on IBM i provided a rich, modern web application interface. This new interface allowed existing IBM i applications access to a non-5250-based interface. This new technology has been instrumental in helping to unlock application on IBM i in a new and modern manner. Many IBM i developers struggled with environments such as Java in the past because of the object-oriented nature of the languages. PHP does not require that sort of paradigm switch. Additionally, PHP encourages you to use the existing back-end business logic that is in your RPG programs. You can use PHP as your modern UI and seamlessly continue to use RPG. Because PHP is a scripting language, it is much easier for the average RPG programmer to pick it up and rapidly become successful with it. There is a large community of PHP developers that can help you seamlessly migrate to writing PHP applications on IBM i. The other reason for having PHP on IBM i is that it opens your IBM i and Power Systems platform to run more new workloads that are based on existing industry PHP applications. For more information about PHP, its importance, and how to quickly and easily get started, see Chapter 12, “PHP” on page 545.

14.6 Ruby: PowerRuby.com

One of the new environments for the IBM i family in the web serving technology area is Ruby on Rails. The PowerRuby.com team has built a production-ready version of Ruby to run on IBM i. Ruby is a rich object-oriented programming language. In many ways, it is not all that different from other object-oriented programming languages. What makes this an attractive web serving alternative is the speed at which Ruby runs coupled with the power and ease of application development that the Rails infrastructure provides. The real power of Ruby is running it on Rails. Rails is the application framework. This delivers the infrastructure, layout, interaction, security, and such so that the application developer can focus their effort on writing application business logic. This framework helps developers get applications to market faster. For more information, see the PowerRuby.com website:

<http://powerRuby.com>

This site provides runtime downloads and more details about running Ruby on Rails on IBM i.

14.7 Conclusion

As you can see, there are many alternatives that are available to you to meet your web serving needs. From pure open source alternatives such as Tomcat PHP or Ruby, to the enterprise function rich WebSphere family, to the ready-at-your-fingertips integrated solutions, IBM i provides a wide number of alternatives to ensure that you keep your applications modern.



HTML, CSS, and JavaScript

Browsers have an enormous role to play in modern applications. They allow developers to create application interfaces without being concerned about the platforms, operating systems, or versions of operating systems that are being used by users.

The major benefit of browsers is that they can make applications available to the world as opposed to only a company's employees or a selective customer base because there is nothing additional to install on the user's device.

Designing a browser interface requires skills that the traditional IBM i developer does not normally have. This does not mean that a traditional IBM i developer cannot write and maintain web pages. It is often the design requirements that can pose a significant challenge. Fortunately, graphic artists or web designers can help provide the design help or you can use one of the many frameworks that are readily available to help.

Even if developers are not going to be actively involved in developing a browser interface, they should have at least a rudimentary understanding of how the browser infrastructure works.

This chapter provides an overview of the programming elements of browser interfaces:

- ▶ The development environment
- ▶ The browser interface
- ▶ HyperText Markup Language (HTML)
- ▶ Cascading Style Sheets (CSS)
- ▶ JavaScript
- ▶ Asynchronous JavaScript and XML (Ajax)
- ▶ The benefit of frameworks

Perhaps the best starting point for those new to web interfaces is the excellent W3Schools website (<http://www.w3schools.com/>), which contains excellent no charge tutorials for web-related programming.

15.1 The development environment

Getting started with web development is easy. All that you need is a PC (any operating system), a browser, and a text editor (there are also many web editors that are available that provide significant productivity improvements over the text editor).

At the time of writing, there are five main browsers:

- ▶ Google Chrome
- ▶ Firefox
- ▶ Microsoft Internet Explorer
- ▶ Safari
- ▶ Opera

In the ever-changing world of the user interface, especially in the mobile market, this list might change by the time you are reading this book.

Although any browser works at the outset, when serious development work starts, developers must ensure that they have adequate testing and debugging tools available. Check on available development tools and ensure that they are adequate before deciding on a browser of choice. Each of the browsers has some form of development tools options available. Some browsers have additional development-related add-ons available, such as Firebug (a JavaScript debugging tool).

Each browser has idiosyncrasies, meaning that a web page might look different when displayed in different browsers. This is especially true of Internet Explorer, which might not only show differences from other browsers but also from older versions of Internet Explorer.

Although a developer can use one browser when developing and testing web pages, it is important that those web pages are tested in the other browsers as well. This is one of the areas in which frameworks can be of benefit.

Although any text editor can be used, it is better to use an editor that does some of the work for you. Developers have many choices for editors, and there are many excellent editors that are available at no charge, such as Kompozer or HTML-Kit. If a developer is also experimenting with PHP, then Zend Studio or NetBeans provide the required editing capabilities.

It might be tempting to invest in one of the expensive design packages. These packages are aimed at designers and often provide many more functions than is covered by the scope of this information.

15.2 The browser interface

Web browsers use a Document Object Model (DOM), as shown in Figure 15-1 on page 615, which is a convention for representing and interacting with objects in HTML, XHTML, and XML documents. The fact that different browsers use different layout engines to represent the DOM is what causes some of the presentation differences between browsers. These presentation differences are one of the issues that HTML5 attempts to address.

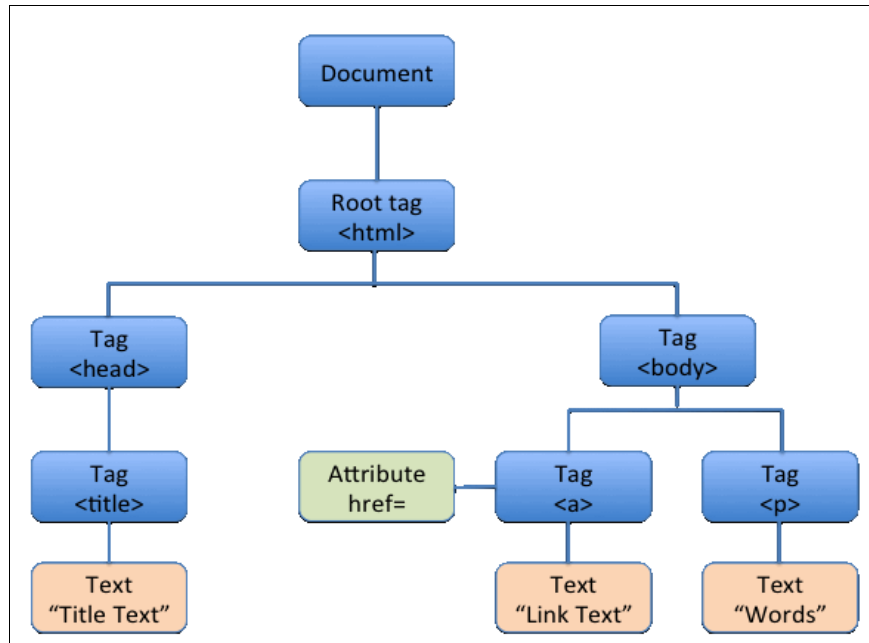


Figure 15-1 The Document Object Model

The tree structure of the DOM means that elements have a parent-child relationship. Usually (but not always), child elements inherit attributes from a parent element. For example, the font to be used in individual paragraphs (<p> elements) is inherited from the <body> element.

A *web page* is a document that defines a DOM tree using some combination of HTML, CSS, and JavaScript:

- ▶ HTML defines the content and structure of the page. Use HTML to define what is in the page and how it is grouped.
- ▶ CSS defines the presentation of the page. It is used to specify the appearance and position of the content. It is used to standardize the appearance and presentation across many pages or even the entire application.
- ▶ JavaScript determines behavior in the DOM, such as reacting to events (for example, clicking an object in the DOM) or manipulating the DOM (dynamically adding new elements to the page or changing the style of an element). HTML and CSS by nature are static. JavaScript provides the logic that is necessary to dynamically update the HTML and CSS for a web page. The JavaScript has direct access to the DOM, which gives it the ability to make these dynamic updates.

Figure 15-2 shows the relationship between HTML, CSS, and JavaScript. As the web has evolved, the distinction and balance between HTML, CSS, and JavaScript has become more defined. The definition of the content and the presentation has become easier, especially with HTML 5 and CSS 3. However, the usage of Ajax and frameworks means that JavaScript has become a major part of programming for the web.

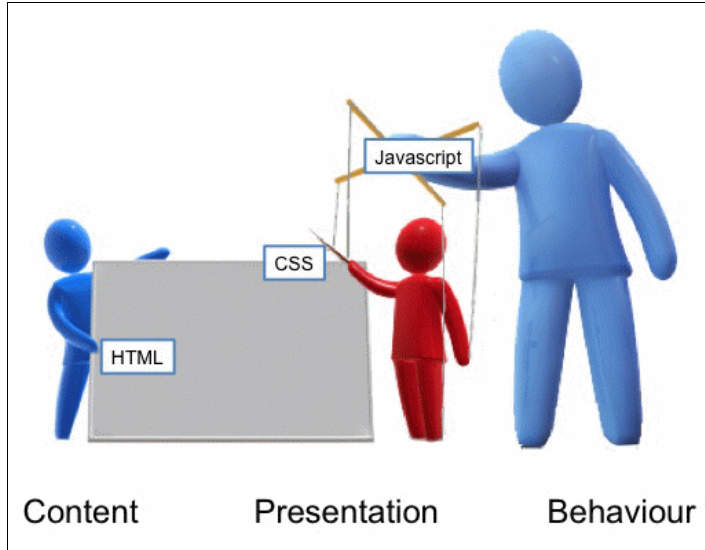


Figure 15-2 HTML, CSS, and JavaScript

It is beyond the scope of this chapter to provide a full introduction to HTML, CSS, and JavaScript, but it is worth looking at some sample code to examine the functions and interaction of the three languages. The following sections examine the HTML, CSS, and JavaScript behind the web page that is shown in Figure 15-3.

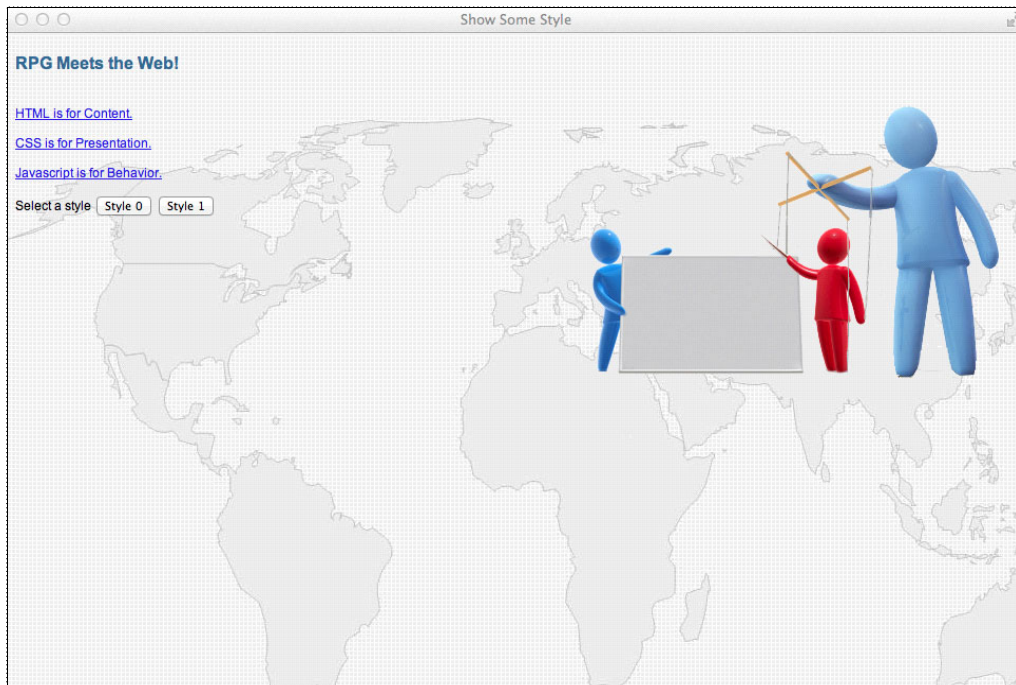


Figure 15-3 A sample web page

15.3 HTML

HTML defines the content and basic structure of a web page. Because HTML is a markup language, it consists of elements that are delineated by tags:

- ▶ HTML is stored in text files with an extension of `.html` or `.htm`.
- ▶ HTML is *not* compiled. It is interpreted and rendered by a browser. A browser always does its best to render a page, even if the code is wrong.
- ▶ HTML tags are keywords that are surrounded by angle brackets.
- ▶ HTML tags come in pairs: an opening and closing tag, such as `<h1>` and `</h1>`.
- ▶ The end tag is written like the start tag, but with a forward slash before the tag name.
- ▶ HTML tags can contain attributes.
- ▶ HTML is static, and requires something like JavaScript to provide dynamic actions or logic.

Example 15-1 shows a simple HTML document that generates the web page that is shown in Figure 15-3 on page 616.

Example 15-1 Sample HTML - a simple HTML document

```
<html>
<head>
  <title>Show Some Style</title>
  <link rel="stylesheet" id="CSSName"
        href="css/css01.css" type="text/css">
  <script type="text/javascript"
        src="js/jsFunctions.js"></script>
</head>
<body>
<div id="container">
  <div class="heading">
    <p>RPG Meets the Web!</p>
  </div>
  <div id="page">
    <p><a href="http://www.w3schools.com/html/default.asp">
      </a>
      <br />
      <a href="http://www.w3schools.com/html/default.asp">
        HTML is for Content.</a>
    </p>
    <p><a href="http://www.w3schools.com/css/default.asp">
      CSS is for Presentation.</a>
    </p>
    <p><a href="http://www.w3schools.com/js/default.asp">
      Javascript is for Behavior.</a>
    </p>

    <p>Select a style
      <button onClick="SwitchStyles(0)">Style 0</button>
      <button onClick="SwitchStyles(1)">Style 1</button>
    </p>
  </div>
</div>
</body>
</html>
```

Here are a few observations about Example 15-1 on page 617:

- ▶ The complete document is contained in an HTML element, between the `<html>` and `</html>` tags.
- ▶ The document consists of two sections: a heading section (`<head>`) and a body section (`<body>`).
- ▶ The heading element (`<head>`) contains elements that are related to the complete document (for example, the JavaScript that is used in the page is often in here).
- ▶ The body element (`<body>`) is the content of the page.

Here are some more basic elements worth noting from Example 15-1 on page 617:

- ▶ The division (`<div>`) elements are used to group or box elements. Divisions are used with CSS to form a consistent layout for the page.
- ▶ The paragraph (`<p>`) element contains text. Parts of the text can be formatted with the bold (``), italics (`<i>`), and underline (`<u>`) elements.
- ▶ The image (``) element is used to place an image in the document. All relevant information about the image is identified with attributes in the `` tag. For example:
 - The `src` attribute identifies the location and name of the image file.
 - The `alt` attribute specifies the text that is shown when the image cannot be displayed.
 - The `title` attribute provides a tooltip when the mouse is left hovering over the image.
- ▶ The anchor (`<a>`) element is used to define a hyperlink.
 - The content of the anchor element is the clickable content in the document. This can be text or an image.
 - The `href` attribute identifies the destination of the hyperlink.

Now that you have seen how to specify the content of a document, it is time to look at how the content can be formatted.

15.4 CSS

Cascading Style Sheets (CSS) are used to control a document's appearance. Style rules can be specified for many things, including the following items:

- ▶ Fonts
- ▶ Text
- ▶ Colors
- ▶ Backgrounds
- ▶ Sizes
- ▶ Borders
- ▶ Spacing
- ▶ Positioning
- ▶ Visual effects
- ▶ Tables
- ▶ Lists

Child and descendant elements usually inherit styles that are defined for parent elements, but there are exceptions to the rule.

CSS can be coded as a style attribute in an HTML tag, in the head section of a document, or in an external document. The preference is to code CSS in external documents (referred to as *style sheets*). In Example 15-1 on page 617, the style sheets that are used in the document are identified by using a link (<link>) element in the heading section.

Before looking at some CSS, there are some points to note in the HTML that is shown in Example 15-1 on page 617:

- ▶ The CSS style sheet `css01.css` is included by using a <link> element in the heading section:

```
<link rel="stylesheet" id="CSSName"
      href="css/css01.css" type="text/css">
```

- ▶ There is a division with a class attribute of heading. Classes are a means of applying style rules to one or more elements.

```
<div class="heading">
```

- ▶ The page contains an image that needs to be formatted and positioned:

```

```

The syntax rules for CSS are simple. CSS code consists of a list of selectors and combinations of properties and values for each selector:

```
selector {property:value; property:value;...}
```

Although the syntax might be simple, the complexity of CSS is in how selectors can be specified and the permutations of properties and values. For example, selectors can be specified for virtually any property of an HTML element. Here are a few examples:

- ▶ HTML tag names
- ▶ Class selectors
- ▶ ID selectors
- ▶ Descendant selectors
- ▶ Selectors that are based on tag attributes
- ▶ Pseudo-classes

Example 15-2 shows the contents of the CSS style sheet `css01.css`. `Css01.css` is a simple CSS style sheet.

Example 15-2 Ccss01.css - a simple CSS style sheet

```
body {background: #EEEEEE;
      font-family:Arial, Helvetica, sans-serif;
      font-size:12px;
      color:#000000;
      line-height:16px;
      background-image: url('../images/worldmap.jpg');
      background-position: top left;
      background-repeat: no-repeat;
      background-attachment: fixed;
}

.heading {font-family:Arial, Helvetica, sans-serif;
         font-size:16px;
         color:#336699;
         margin-bottom:0px;
         margin-top:20px;
         font-weight:600;
```

```
}  
  
img {float: right;  
    padding-left: 10px;  
    padding-right: 5px;  
    padding-top: 5px;  
    border: 0;  
}
```

This example contains directives for the following selectors:

- ▶ The selector for the body identifies properties for the <body> element in a document, including values that relate to font, font size, and color in addition to defining a background image for the page. This ensures that everything on the page is displayed with the exact same settings for a consistent look.
- ▶ The selector for .header is the definition of a class selector. Class selectors start with a period. Values are specified for properties that are related to font, font size, and spacing. In this example, elements in the document normally have the values that are specified for <body> unless the element is assigned a class of header. Because of inheritance, all elements on the page use these values for font, color, and positioning.
- ▶ The selector for img identifies properties for any element in a document, including values for padding around an image and where it is positioned on the page.

Although this is a simple example of a style sheet, it should suffice to demonstrate that the style of a page is determined by CSS as opposed to HTML. This is a theme that is examined in more detail after a quick look at JavaScript.

15.5 JavaScript

JavaScript is a lightweight interactive programming language that is automatically enabled in every browser. JavaScript can be coded directly in an HTML document or can be coded in a separate document (the preferred method) that is imported when the document is loaded. The file that contains JavaScript is identified by the .src attribute in a script tag. The purpose of JavaScript is to dynamically manipulate the HTML and CSS for web page. Here is an example of JavaScript:

```
<script type="text/javascript"  
    src="js/jsFunctions.js"></script>
```

JavaScript can be used to perform the following functions:

- ▶ Put dynamic text in a page.
- ▶ Read and change the content and attributes of an HTML element.
- ▶ Validate data and signal errors (before it is sent to a server).
- ▶ Manage cookies.
- ▶ React to events.
- ▶ Manage an Ajax conversation with a server.
- ▶ Change CSS appearance, style, color, and so on.
- ▶ Do many more things.

These are some of the basic rules governing JavaScript:

- ▶ All code and variable names are case-sensitive.
- ▶ Statements end with a semicolon.

- ▶ Code is blocked with curly brackets {}.
- ▶ Single line or end of line comments start with //.
- ▶ Multiple line comments can be placed between /* and */.
- ▶ Variables are not typed and can be declared anywhere within the code with the var statement.
- ▶ Special objects are required for arrays and dates.
- ▶ Values can be assigned at declaration.
- ▶ JavaScript uses standard operators in expressions and has standard operation codes for if, switch, for, while and do while operations.
- ▶ JavaScript can be either procedural or object-oriented.

To demonstrate JavaScript at work, we look at a function that allows the presentation of a page to be dynamically changed by clicking a button. The presentation style can be alternated between the styles. The original web page that is shown in Figure 15-3 on page 616 has two buttons that are labeled style0 and style1. Clicking the **style0** button causes the layout to switch to the layout that is shown in Figure 15-4.

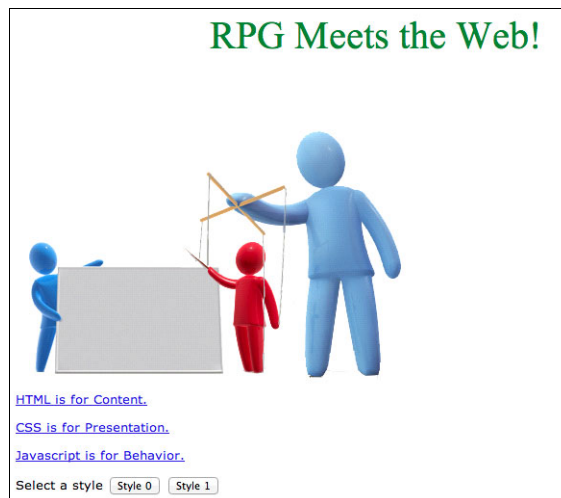


Figure 15-4 Changed presentation of a “Sample Web Page”

This is a simple example of how the presentation style of a page can be changed by simply applying different CSS rules. When a button is clicked, a JavaScript function changes the name of the CSS style sheet to be used for presenting the page. Remember, the CSS style sheet is identified with the href attribute in a link tag as follows:

```
<link rel="stylesheet" id="CSSName"
      href="css/css01.css" type="text/css">
```

The JavaScript function changes the value of the href attribute. The link element has a unique ID of CSSName.

Example 15-3 shows the code for the function **SwitchStyles()** in the JavaScript file `jsFunctions.js`.

Example 15-3 jsFunctions.js - a simple JavaScript function

```
function SwitchStyles(newStyle) {
    sheetNames = new Array("css00.css", "css01.css");
```

```
document.getElementById('CSSName').href =  
    "css/" + sheetNames[newStyle];
```

Here are the features of the function **SwitchStyles()**:

- ▶ The function accepts a single parameter (*newStyle*), which is a number (0 or 1).
- ▶ An array (*sheetNames*) is defined with elements that contain the names of the different style sheets that can be used.
- ▶ The href attribute of the element with an ID of *CCSName* (that is, the link element in the document) is changed to the name of the style sheet in element *newStyle* of the array *sheetNames*. In JavaScript, the first element of an array has an index of 0.

The simplest way to call the function is to associate the function with an event on the two buttons. The event of interest is when the button is clicked, so use the `onClick` attribute is used to specify the function to be called and the parameter to be passed when the button is clicked, as follows:

```
<button onClick="SwitchStyles(0)">Style 0</button>  
<button onClick="SwitchStyles(1)">Style 1</button>
```

For completeness, Example 15-4 shows the contents of the style sheet `css00.css`.

Example 15-4 Ccss00.css - another simple CSS style sheet

```
body {background: #BABFFA;  
    text-align: center;  
}  
#container {background: #ffffff;  
    position: relative;  
    width: 750px;  
    height: 1000px;  
    margin-left: auto;  
    margin-right: auto;  
    text-align: left;  
    font-family: "Verdana", sans-serif;  
    font-weight: 400;  
    font-size: 13.0px;  
    padding-left: 10px;  
    padding-right: 10px;  
}  
.heading {font-family:"Times New Roman", serif;  
    font-size: 40.0px;  
    line-height: 1.20em;  
    text-align: center;  
    color: #008000;  
    padding-top: 20px;  
    padding-bottom: 10px;  
}
```

15.6 Ajax

Ajax is a means whereby a browser scripting language can be used to asynchronously transmit small amounts of information to and from the server. This means that, in the background, a browser can communicate with a server while a user is interacting with the page. This approach is fast becoming the norm in web design where the tendency is to initially download a minimum amount of data to the browser and retrieve more information only when required. Although this approach might be considered a nice design technique for PC-based browsers working over a fast network, it is a necessity for browsers on mobile devices. Mobile devices are often dependent on telephone networks, which do not offer the same download speed; in such cases, a smaller amount of data is better.

Consider the following aspects of Ajax:

- ▶ Ajax is not a programming language; it is an approach to web interaction.
- ▶ Although the ability to have asynchronous communication with a server has been around for a long time, the term Ajax was not coined until the Internet achieved high speeds, PCs became faster, browsers became better, and some work was done by Google. Google pioneered the usage of Ajax, especially with the introduction of Google Suggest and Google Maps.
- ▶ Ajax needs only to be coded. All the major browsers support it, so there is nothing to install or activate. You need only to use it.
- ▶ Although Ajax was originally considered to be an advanced programming technique, the introduction of frameworks has made Ajax much easier to use.
- ▶ In most browsers, Ajax means using an XMLHttpRequest object to communicate with the server. In different versions of Internet Explorer, this object has different names.
- ▶ The information that is transmitted to and from the server can be in any format. The original intention was that XML would be used, hence the X in Ajax. However, XML has proven to be verbose for the intended usage of Ajax (small amounts of data), so JavaScript Object Notation (JSON) is more commonly used.

Learning to use the XMLHttpRequest object (and the Internet Explorer alternatives) can be challenging, so the easiest way to master Ajax is to use frameworks. For more information, see 15.8, “The benefit of frameworks” on page 631.

15.7 HTML5 and CSS3

HTML5 and CSS3 are the latest trends in web development. As of this writing, both standards are still in active development. This does not mean that you should avoid using their features. Just as web development techniques are constantly improving, web standards are now becoming more agile. In fact, the HTML5 and CSS3 standards might never be finalized, and that is a good thing.

15.7.1 HTML5

Many different technologies are commonly referred to generically as HTML5, even though they are separate technologies with their own standards. For an excellent illustration of all the related standards and technologies that are brought together in HTML 5, see the following website:

<http://commons.wikimedia.org/wiki/File:HTML5-APIs-and-related-technologies-by-Sergiy-Mavrody.png>

You probably noticed that there are two HTML5 standards. These two separate HTML5 standards are created by two different organizations:

- ▶ World Wide Web Consortium (W3C)

The W3C is the older of the two standards organizations. It was formed in 1994 at the Massachusetts Institute of Technology Laboratory for Computer Science by Tim Berners-Lee. The W3C standard can be found at <http://www.w3c.org/TR/html5/>.

- ▶ Web Hypertext Application Technology Working Group (WHATWG)

The WHATWG was formed in 2004 by individuals from Mozilla, Opera, and Apple after the W3C rejected a position paper from Mozilla and Opera on Web Forms 2.0. Web Forms that later developed into the HTML5 standard. The WHATWG standard is described at <http://www.whatwg.org/html5/>.

Although the two standards agree on most topics, there are a few areas where they diverge. These differences are beyond the scope of this book, but for most modernization work the differences do not come into play.

Now that you know a bit about what HTML5 is and its origins, it is important to understand what makes it important. HTML5 is an attempt to standardize many of the commonly used web development techniques. The last HTML standard, HTML 4.01, from the W3C was finalized in 1999. So as of this writing, the HTML standard is over 13 years old. Stop and think about how long that is in terms of technology.

Even though the HTML standard went unchanged for that long, the content of the web certainly continued to progress. This unstandardized evolution of web development techniques led to variations in the usage and implementation of many of the technologies that are indispensable to any modern web application. Any person who has spent time doing web development can tell you about browser compatibility issues. Each browser has its own method of implementing some web technologies leading to complicated HTML and JavaScript code to ensure consistent results. The HTML5 standard attempts to take commonly used web techniques and up-and-coming technologies and standardize them to make them easier to use. Common web application tasks that formerly required extensive code or third-party plug-ins are now natively implemented by using standard HTML. Things such as animation, embedded audio or video, storing data locally, form field validations, and geolocation services are now easily implemented with a few simple lines of code. Example 15-5 on page 625 demonstrates the simplicity of embedding a video in HTML5.

```
<html>
  <body>

    <video width="640" height="480" controls>
      <source src="movie.mp4" type="video/mp4">
      <source src="movie.ogv" type="video/ogg">
      This browser does not support HTML5 video.
    </video>

  </body>

</html>
```

In this simple example, the page contains a streaming video. Notice that all you need to do is specify the new `<video>` tag. The example sets the width and height in pixels and also has the controls option specified. The controls option enables controls for starting and stopping the video. The next two lines list two possible versions of the video. The great thing about the video tag is that all the major browsers support it in their latest versions. The issue is that not all browsers support the same video file formats. By using the `<source>` tag, the page can list as many versions of the file as are available and the browser chooses the first one that it can use. The current recommendation is to include MP4 and OGG versions of videos because all major browsers can support at least one of those formats. The text inside the `<video>` tag is displayed only if the browser does not support HTML5 video (for example, certain older browsers).

Mobile

Although all of the major desktop browsers now support many of the features of HTML5 in their newest versions, one of the biggest catalysts for HTML5 is the explosion of the mobile device market. Mobile devices redefine the expectations of users regarding web applications. Because of their advanced hardware and always on nature, these devices require new functions that the current web standards cannot handle. This paradigm shift to mobile technology has kept mobile browsers on the bleeding edge of supporting proposed technologies in the HTML5 standard.

Here is one scenario to consider. One of the challenges of mobile devices is that they are mobile. This means that a constant network connection cannot be assumed. So if you had a simple mobile web application to allow store managers to place orders for your products, how might you allow for them to continue entering orders if they lose their WiFi or cellular connection? HTML5 makes this scenario fairly easy to deal with. Here are some of the technologies in HTML5 that are useful in this situation and why.

Application Cache feature

The HTML5 Application Cache feature allows you to create a manifest file with a list of files that are required for your application to function when offline. The manifest file should list all HTML, CSS, JavaScript, images, and any other type of files that are required. Example 15-6 shows a simple manifest file.

Example 15-6 Application cache manifest file

CACHE MANIFEST

```
CACHE:
index.html
my_logo.png
my_styles.css
my_scripts.js
```

The manifest file is linked to the application by adding it to the <html> tag as follows:

```
<html manifest="myapp.manifest">
```

Now that your web application can continue to function when offline, it needs the ability to detect and react to changes as its connection status changes. HTML5 provides the `navigator.onLine` property that can be checked to find out whether the application is online. To handle changes in the application's connection status, the `online` and `offline` events fire (are called) when a connection is established or lost, respectively. Example 15-7 shows the code to detect connection status changes.

Example 15-7 Detecting connection status using JavaScript

```
// Using navigator.onLine to detect status
if (navigator.onLine) {
    // App is online
} else {
    // App is offline
}

// Handling connection status changes
document.addEventListener('online', goOnline);
document.addEventListener('offline', goOffline);
```

The first section of Example 15-7 demonstrates how you can check the connection status on demand. The `navigator.onLine` property is automatically maintained by the browser and can be checked at any time. In the second section, the example adds event handlers for the `online` and `offline` events. The first line adds the handler for the `online` event. What that means is that anytime the connection status changes to `online`, the browser automatically calls the `goOnline` function. The second line makes the browser call the `goOffline` function whenever the connection goes offline.

These handler functions can be used to change the behavior of the application based on the connection status. In this scenario, when orders are entered and the device is offline, the application should store the order data locally. When the device reestablishes its connection, it should send any locally stored data to the server-side program on the web server and begin sending any new order data as entered.

Web storage

Section “Application Cache feature” on page 626 covered how to make your application run offline and change behaviors based on the connection status. The proposed solution is to store the order data locally when offline. HTML5 also has that covered. The secret is web storage.

Traditionally, if a web application needed to store data locally on the user's machine, it did so by using a *cookie*. Cookies have a few issues. They are limited to a size of 4 KB. They also have been linked to numerous security vulnerabilities over the years. Security issues aside, one of the biggest problems with cookies that are used to store large amounts of data is that the cookie data is carried along on the HTTP request. So, if you stored 2 KB of data in a cookie, that 2 KB is tacked on to all traffic going to and coming back from the web server. In the world of mobile devices with limited bandwidth and metered data plans, this is unacceptable.

Web storage is a huge improvement compared to cookies. It is never transmitted back to the server, saving bandwidth. Data is stored in simple key/value pairs. Web pages can access only their own storage, which alleviates many security concerns. The storage size is also larger. Although the web storage standard does not specify a specific maximum storage size, the major browsers seem to have all settled on a maximum size of 5 MB.

There are two forms of web storage:

- ▶ `localStorage`

Data that is stored by using the `localStorage` object is stored indefinitely. If the user leaves your web application and comes back at a later date, your `localStorage` data is still available. For this reason, it is important that your web application cleans up after itself by removing items in `localStorage` when they are no longer needed.

- ▶ `sessionStorage`

The `sessionStorage` object functions identically to the `localStorage` object with one major difference. When the user's session ends, the data that is stored in `sessionStorage` is automatically wiped out. This behavior is helpful if your application does not require data retention between sessions because you do not need to worry about cleaning up the storage that is used by your application.

Best of all, using web storage is simple. Example 15-8 demonstrates how to interact with data in the `localStorage` object.

Example 15-8 Using web storage in JavaScript

```
// checking to see whether localStorage is supported
if (localStorage) {
    // browser supports webstorage
} else {
    // not supported
}

// saving data in localStorage
localStorage.setItem('myKey','myData');
localStorage.myOtherKey = 'myOtherData';

// retrieving data in localStorage
myVar = localStorage.getItem('myKey');
myOtherVar = localStorage.myOtherKey;

// deleting a single value
```

```
localStorage.removeItem('myKey');

// clearing out all of my local storage
localStorage.clear();
```

The same methods are also used when dealing with the sessionStorage object.

As you can see, HTML5 provides some significant features. These examples are only a few examples of the functions that are built into HTML5. There are many other features that can significantly help applications that are running in a mobile environment. All of the features that are mentioned here for mobile development are also available in desktop environments.

15.7.2 CSS3

CSS3 is the latest set of standards for CSS. With the introduction of CSS3, the CSS language was broken into modules. These modules are independent standards that can change version independently. Why is that important? It is important because now new functions can be introduced in any of the individual modules whenever they are ready. Features no longer must wait for enough content for a new version of the entire CSS language. When an individual module is ready for release, it can be released to the user community.

CSS3 introduces too many features to cover all of them in this chapter. There are many excellent books and online resources that are available to bring your skills up to speed. Instead, this chapter touches on a few of the interesting additions.

Borders

One of the most useful additions to CSS3 is the ability to style boxes with rounded corners and shadows using nothing but CSS. In the past, to have a rounded box, you had to design the box using a graphical editing program such as Adobe Photoshop or the open source program GIMP. Then, you had to slice the box into corners and sides and save images of each to be used in your application to create the graphical box. This was tiresome and required additional skills and tools. In CSS3, the same thing can be accomplished by using a few lines of code. Figure 15-5 shows what a text box with rounded corners and shadowing looks using CSS3 support.

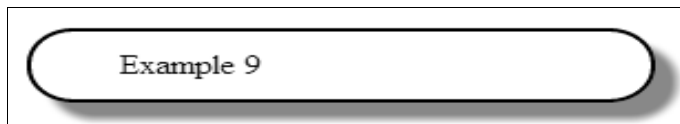


Figure 15-5 Figure 6. CSS3 Rounded corners and shadows

Example 15-9 shows the CSS definition for the <div> tag. In this definition, we specify a 2px wide border with rounded corners and a shadow. This is the CSS code that is used to create Figure 15-5.

Example 15-9 Rounded and shadowed box in CSS3

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      div
      {
        border: 2px solid;
```



```

        -webkit-box-shadow: -5px -5px 5px #888;
        box-shadow: -5px -5px 5px #888;
    }
</style>
</head>
<body>

    <div class="example10">
        <div class="transition">IBM i</div>
    </div>

</body>
</html>

```

Take the code above and copy it into a file on your desktop. Call it `animate.html` and open it with your browser. You see the animation when you move your mouse over the blue square. Figure 15-6 shows how the blue box tumbles and then stops with shadowing around it.

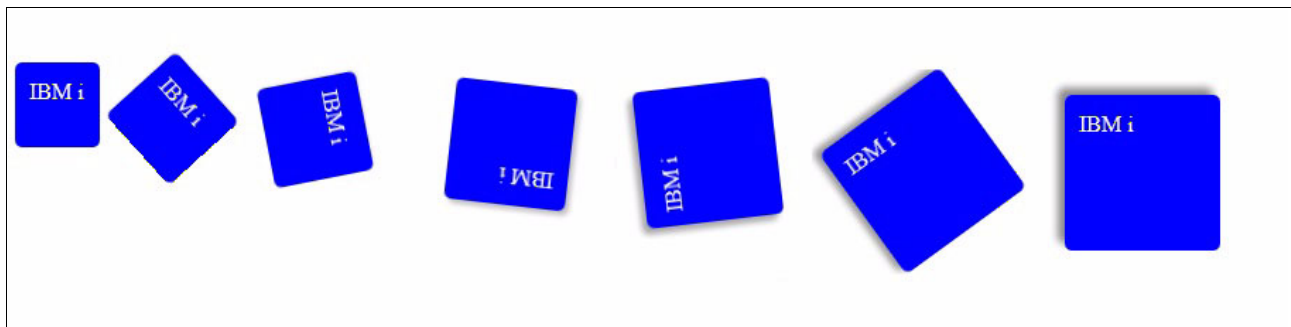


Figure 15-6 Animating blue box with shadowing

The CSS is not as complex as it seems. If you examine it, you notice that much of it is the same code that is repeated with different prefixes, such as `-webkit-`, `-ms-`, `-o-`, or `-moz-`. These are necessary for the example to work in different browsers, that is, Google Chrome, Internet Explorer, Opera, and Firefox respectively.

The `div.example10` section defines the outer `<div>` in our example. The next section, `div.example10 div.transition`, defines the initial state of our blue box `<div>` and how long the transition should take. In this case, the transition, or animation, should last three seconds. The last section, `div.example10:hover div.transition`, defines the final size, shadow, and animation of the blue box `<div>` after being moused over. The blue box increases to 90 pixels square. The left margin moves to 300 pixels. The box rotates 360 degrees, and it is shadowed.

Media queries

Media queries are a way to condition CSS code that is based on information about the output device. This capability is key to *responsive web design*. Responsive web design aims to provide optimal viewing experiences across many different devices, which ensures that your web application formats correctly on a smartphone, tablet, desktop, or any other type of device. To accomplish this task, CSS3 Media Queries allow you to change CSS properties based on the characteristics of the device. Example 15-11 on page 631 shows an example of how this is set up.

Example 15-11 CSS3 media queries

```
div.banner {  
    width: 900px;  
}  
@media (max-width: 480px) {  
    div.banner{  
        width: 480px;  
    }  
}
```

In Example 15-11, the first section defines any CSS rules for the `<div>` with a class of `banner` on the page. The second section is a media query. The `@media` denotes a media query. In this case, it is conditioning on a screen width of 480 pixels or less. When the query is true, the example changes the width of the `<div>` tag with class `banner` to 480 pixels instead of 900.

This is a simplistic example. However, you can override as many rules for as many media queries as you want. This capability allows you to have different CSS for different devices to ensure that your application looks good on any supported device.

Again, there are many examples for CSS3, but this book is intended to provide a brief explanation of what is available. There are many books and websites that can help you learn more about helpful features of CSS3.

15.8 The benefit of frameworks

Programmers tend to want to be in total control of what they code. Unfortunately, this can be a frustrating tendency when it comes to programming web interfaces.

Coding for the web is full of idiosyncrasies. Each of the browsers has its own quirks and some of them even have quirks between versions of the same browser.

Although every programmer should have a good knowledge of how to code, they also need to know where to draw the line and ensure that they are not reinventing the wheel.

The good news is that there are a myriad of frameworks that are available to help with constructing HTML documents and CSS style sheets and to simplify the coding of JavaScript to manipulate the DOM.

The bad news is that there are a myriad of frameworks available. The world of web development is evolving at a rapid pace and new frameworks come into existence every week. Some experimentation is required to find one that works best for you. The key to using any framework is to ensure that it is easy to use and, if necessary, easy to replace.

For the design of web pages (HTML and CSS), see the following websites:

- ▶ HTML5 Boilerplate
<http://html5boilerplate.com/>
- ▶ YAML 4
<http://www.yaml.de/>
- ▶ Skeleton
<http://www.getskeleton.com/>

- ▶ Foundation
<http://foundation.zurb.com/>
- ▶ Bootstrap
<http://getbootstrap.com/2.3.2/>

There are also many frameworks that are available for JavaScript. At the time of writing, here are the two main contenders:

- ▶ jQuery
<http://jquery.com/>
- ▶ ExtJS
<http://www.sencha.com/products/extjs>

A simple search provides a list of numerous frameworks for you to consider.

15.9 Going to the web

The web is not quite as daunting as it might appear at first. It is a fast-changing environment and is full of frustrating foibles, but it also provides a rich interface that allows for innovative user interaction. It is not that difficult to code, especially when you use frameworks.

You do not need to be a graphic designer to code for the web. You can use someone else's design and code.

Get some tutorials and learn HTML, CSS, and JavaScript, familiarize yourself with some frameworks, and start coding.



Security

As with any project, security planning should be a major part of a modernization project. Security should be considered for all changes that are made when you modernize, but modernization is a great time to examine the overall security of your application and your system. There are multiple areas, or levels, to consider regarding security.

This chapter covers some basic concepts. There are entire books on the topic of IBM i security and companies that specialize in securing systems. A great place to start is *Security Guide for IBM i V6.1*, SG24-7680.

This book and some web searches provide the information that you need to better understand and start implementing these concepts.

16.1 Introduction

The topic of security is often avoided by application architects and developers. It is often viewed as something that impedes progress. Rarely is anything positive said about security in the application development world. However, as times have changed for application and user interface design, times have also changed for protecting an organization's data. Today's world requires that you consider the security requirements of the data that you are working with. Thus, the inclusion of this chapter.

Many laws and regulations now govern how data must be protected. This is not just a US-based issue. Laws dictating how to handle private data and financial data span the globe. In addition, more organizations are realizing the value of the data that is stored on their IBM i systems and are looking to ensure that it is appropriately protected. How do you determine how to protect the data of the specific application on which you are working? It depends on the type of the data. By type, we are not talking about character data or signed integers. The type of data in this case is whether the data should be considered private data, such as credit card, financial, healthcare, or social security and social insurance numbers. Some of this data comes under direct control of specific laws or regulations, so the implementation of the security requirements is dictated by those laws. Other data, such as data that is unique to the organization (such as vendor lists, pricing, and sales figures) might also need to be secured. You must work with the data owner to determine the legal and organization-specific requirements for protecting the data. All of the topics that are described in this chapter are intended to help you reduce risk to the data.

16.2 Classifying data

The data that your application works with is probably classified by your organization as confidential, private, or internal use only (or something similar). The classification of the data is important because it defines the authority settings on the database file and the following items:

- ▶ Data encryption

Whether the data must be encrypted, either during the transmission of data (for example, during an ODBC or JDBC connection) or when it is at rest (for example, stored in a database file, data area, or user space).

- ▶ Data retention period

Financial data often must be retained for seven years. However, privacy regulations demand that data that does not need to be retained and should be removed as soon as possible, or that the data must be de-personalized so it can no longer be associated with the individual. This means that you might have to have a mechanism to support this data retention period.

- ▶ Additional authentication requirements

Authentication, or proving that a user is who they claim they are, is done when the user starts the application. However, some data is so sensitive that some organizations require an additional form of authentication before accessing that data. An example is a healthcare organization that requires a finger print to authenticate to a list of users who are approved to see the medical records of high-profile individuals.

- ▶ Auditing or logging

Some types of data might require more auditing than what is provided by the operating system. It must be provided in the application to ensure that a complete picture of who accessed or changed data can be assembled. This ensures that incidents can be investigated thoroughly and fraud detection can be implemented.

16.3 User authentication

If you are writing a web- or client-based application, you must perform user authentication. (Native IBM i applications do not need to do this authentication because the operating system has done the authentication for you.) The method that you use for authentication depends on whether the application is going to be used internally or externally available as an Internet application. Several options exist. This section examines the benefits and limitations of each option.

16.3.1 IBM i user profile

This method is best suited for intranet applications. Users likely have an IBM i user profile, so using this method as an authentication method means no additional work for the system administrators.

This method has the following benefits:

- ▶ It allows you to use the IBM i commands and utilities that come with the operating system to manage the application users.
- ▶ Logging of who reads or changes an application database file is performed through the system that uses journal and database journals. Journal objects are inherently secure, so no additional protection of the journal receivers (the objects that retain the log information) is required.

This method has the following limitations:

- ▶ Unless you can control browser or client configuration for application users (which is possible if the application is only used internally), credentials are cached in the browser until the browser or client application is closed.
- ▶ If you are using this application as an externally facing internet application, administrators likely must maintain thousands of additional user profiles to manage access to the application.
- ▶ Because each application user has a corresponding user profile, if they can somehow gain access to the system (for example, through a terminal emulator or an FTP or ODBC session), they can sign on to the system and run as any normal user of the system.

16.3.2 Validation list user

A validation list user, sometimes called an “Internet user”, is named after the IBM i object that enables the validation list user function. A validation list allows you to register users for use with a web (or other) application. This method of authentication is best suited for internet applications or applications where you do not want the application user to have to have an actual IBM i user profile.

The validation list contains the user's identifier and authentication information (typically a password). The Apache web server on IBM i has a configuration option to use a validation list as the user repository for a web application. When a user starts the web application and is challenged by the browser to authenticate, the validation list that is specified in the Apache configuration is checked for the valid user ID and password combination.

This approach has the following benefits:

- ▶ Web application users do not have an IBM i user profile. Therefore, the credentials that are used to authenticate to the web application cannot be used to access the system directly.
- ▶ For web applications, all authentication is done by the Apache web server. The Apache web instance calls the validation list APIs to perform the authentication.
- ▶ Authentication information is stored encrypted. You do not need to create a secure method for storing this information. APIs are available to authenticate validation users.

This method has the following limitations:

- ▶ Unless you can control browser and client configuration for application users, credentials might be cached.
- ▶ You must develop your own method of managing these users. Although the APIs are provided by the IBM i operating system, a method to manage the entries in the list (that is, validation list users) is not. You must manage the password composition rules and password expiration interval and determine which users are inactive and disable them (either remove the password or the whole entry) yourself.
- ▶ Logging of who read or changed an application database must be done by the application. The integrated logging features of IBM i log all of the changes as being performed by the profile that is running the HTTP server (if you use the default setting, it is the profile QTMHHTTP1) or the profile that is making the connection from the client to the server. This process is not sufficient for many compliance requirements.

If you choose the validation list option, consider the following things:

- ▶ When writing a web application, consider changing the profile under which this Apache web instance runs. The default profile is the IBM supplied profile QTMHHTTP1. Some vendor products grant authority to this profile so their GUI interface can access the product's libraries.
- ▶ Limit what can be accessed by the application profile. It should be created as Initial pgm *NONE, Initial menu *SIGNOFF, Limited capability *YES, No special authorities, and so on.
- ▶ Consider excluding the application profile from other application libraries and directories that this application does not need to access.

16.3.3 LDAP

Many organizations already have an LDAP directory for managing authentication for various applications within their enterprise. One of the options that are provided by the Apache web server is to authenticate users using LDAP. This method has many of the same benefits and limitations as the validation list user method, with the exception that your organization might already have tools and processes in place to manage these users. Make the same considerations for the LDAP option as are made for the validation list user option.

16.3.4 Application-defined authentication

Another method of authentication to consider requires the application itself to perform the authentication (instead of having the web server perform the authentication, for example).

This method has the following benefit: For web applications, there is no danger of credentials being stored in a browser session because the authentication is no longer performed by the browser/web server.

This method has the following limitation: In addition to the limitations of the validation list user, you must develop all of your own methods to create and manage the application users, including the secure storage of credentials. No APIs are available for this process unless you use validation list users.

16.3.5 Kerberos

Instead of using a user ID and password to authenticate, you can use a Kerberos ticket to authenticate the user. This method requires that there be a Kerberos server somewhere in the network (likely tied to your LDAP server or Active Directory); therefore, it is suited only to an intranet application. If single-sign on (SSO) is implemented or is an objective for your organization, authenticating through a Kerberos ticket means that your application fits right in.

This method has the following benefit: Management of passwords is eliminated and your application can participate in SSO.

This method has the following limitation: If Kerberos authentication is being used in your organization (and it is a Microsoft network), you need to write only to the appropriate APIs to authenticate the user. If a Kerberos server is not in place, then you must get one running first.

For more information, see the Network Authentication Service and Enterprise Identity Mapping (EIM) topics and APIs in the IBM i and System i information center:

<http://publib.boulder.ibm.com/eserver/ibmi.html>

16.3.6 *ALLOBJ special authority

Whatever profile you decide to have access the application data on IBM i, it does not need *ALLOBJ special authority. It needs authority to the data that is being accessed, but it does not need authority to everything on the system, which is what *ALLOBJ provides. Requiring application users to have *ALLOBJ or creating the application profile accessing the data with *ALLOBJ can cause problems for your administrator during an audit. Worse, it creates risk to the data that is unnecessary. Your goal should be to require only enough authority to run your application and no more. This is referred to as *least privilege access*. Again, some laws and regulations require it. Be a responsible developer and do not require *ALLOBJ. More information about the options for obtaining authority to data are provided in 16.3.12, “Authority settings on the data” on page 640.

16.3.7 Application owner profile

Your application objects that are on IBM i must be owned by an IBM i profile. Create your own user profile to own (and potentially run) your application. Although it is easy to use an IBM-supplied profile such as QPGMR or QTMHTTP1 to own your application, it is not a preferred practice. Many application developers have already done this and have “overloaded” the usage of these profiles. It can be hard to accomplish separation of duties between applications when they are all owned by the same profile. Also, log or audit entries can become ineffective if all access is performed as the same profile. It is also difficult to know what objects (especially programs) belong to the operating system and what objects belong to the application, which can present challenges when upgrading and for back-up and recovery, high availability (HA), and disaster recovery processes. Finally, some organizations have added special authorities to profiles such as QPGMR and QSYSOPR. In these situations, you might unknowingly run your application with a profile that has more authority than you intended and security checks that you have implemented might no longer be effective.

Create your own profile with no special authorities, no group profiles, limited capability-*YES, initial program-*NONE, and initial menu-*SIGNOFF so it cannot be used to sign on to an emulation session. If it is not being used to make a connection to IBM i, set the password to *NONE.

16.3.8 Connection profiles

This term varies. Some organizations call these profiles application or service profiles. Whatever they are called, the profiles that are described in this section are the ones that make a connection from a client (using ODBC or JDBC) or a mobile device to IBM i. These profiles must authenticate to IBM i, so they typically require a hardcoded password. An alternative is to use an identity token instead. (For more information, see the “Identity Token” topic in the IBM i information center for more information.) If you must use a hardcoded password, assign a strong password, store the password encrypted, and do not make the source code that retrieves the password available to the entire development community.

If you have more than one application that makes these types of connections, create a separate profile for each application. This makes debugging easier and audit entries more meaningful. Create the profile with no special authorities, no group profiles, limited capability-*YES, initial program-*NONE, and initial menu-*SIGNOFF so it cannot be used to sign on to an emulation session. You probably set the password expiration interval to *NOMAX, but understand that your security administrator and auditor likely require you to change this password at least annually.

16.3.9 Logging

Any time that an application is written such that the profile that accesses the database on IBM i is always the same profile (such as when the IBM i is the back-end data store and the application interface is on another server or a web application with internet users), you might have to add additional logging to your application. IBM i has robust logging capabilities through the combination of the audit journal and the database journal. When the database is accessed with only the application or web server profile, the ability to detect fraud or determine which user is performing the application task is restricted. To overcome this restriction, you might have to add additional logging to your application. If this is a requirement, be sure to log at least the date and time of the transaction, the application user, and the action being performed. Work with the business to make sure that you are logging everything that they require.

In addition, you must secure this log file. Have the file be owned by the application profile, set the *PUBLIC authority to *EXCLUDE, and write to it by using a program that adopts the application profile (that is, the user profile parameter of the application is set to *OWNER). For legal and audit purposes, this file must be protected.

16.3.10 Separation of duties

When writing an application, consider whether you must provide levels of access that allow your organization to have separation of duties. The level of access typically corresponds to the job position or department to which an individual belongs and is often called *role-based access* (RBAC). You must consider providing RBAC if the application has situations where some users should be allowed to perform specific functions and others should not be allowed. In this case, you must provide a mechanism that allows an administrator to specify whether a user or a group of users is allowed to perform the task. IBM i has “function usage” APIs that allow you to define and register your own application functions. Administrators can then administer access through “green screen” commands or through Application Administration (a feature of iNavigator).

16.3.11 Encryption

Although some regulations require it, more organizations are choosing to encrypt data because of the privacy issues and the threat of data getting into the wrong hands. If the data is encrypted and the encryption key is not available, the data is useless. Here are some of the key considerations:

- ▶ If you are passing authentication information (for example, a user ID and password), it must flow over an encrypted session, such as SSL-enabled ODBC, JDBC, or HTTP request.
- ▶ If the data that is being collected is classified as private data or data that is confidential to the organization, it must be transmitted over an encrypted session.
- ▶ Private data must be stored in encrypted form, with controls in place as to who can de-encrypt the data. Version 7 Release 1 can use the SQL field proc function to automatically encrypt and decrypt data in a specific column. However, you must add some sort of access control mechanism to this function because from a security perspective it does not add anything if you blindly decrypt the data for all accesses. In this case, why bother encrypting the data? Because of the ease of adding this function to a database (as compared to reworking an entire application's structure to encrypt an existing field or to add an encrypted field as was previously required), many organizations are choosing to encrypt data that is not legally required to be encrypted. Work with the data owner to determine whether this is a requirement.
- ▶ The encryption key must be kept private, not the encryption algorithm. Do not attempt to write your own encryption algorithm. Use a well-known algorithm.
- ▶ Key management is important. The key must be kept secure and only available to selected administrators, and mechanisms must be available to decrypt and reencrypt the data if the key is compromised, an administrator that knows the key leaves the organization, or your security policy states that keys must be changed regularly. Consider using a vendor product that has worked out these processes and focus your efforts elsewhere.

16.3.12 Authority settings on the data

It is important to plan authority settings on the data and have sufficient authority to access and update application data. The determination of what authority is required starts with how the data is going to be used by your application. The determination of where that authority is going to come from starts with the default authority on the database file, which is the *PUBLIC authority setting.

The *PUBLIC authority setting must reflect the business' requirement for default access. Some organizations require all confidential and private data to be “deny by default”, that is, the *PUBLIC authority of *EXCLUDE. Other organizations have a default access policy of read-only access for all of their data, that is, *PUBLIC authority set to *USE.

If your application is simply reading data and performing no updates and the *PUBLIC authority is *USE, then no additional authorities, either from the profile accessing the data or through application programs, are required. However, it is rarely that easy.

Consider a scenario where the organization requires data to deny by default (*PUBLIC authority is *EXCLUDE) and your application is reading, copying, updating, and deleting records in the application database file. Here are some options for obtaining authority:

- ▶ Adopted authority

You can get much more complex, but the easiest way to ensure that the application has sufficient authority to access the database file without authorization failures is to use adopted authority. Have the application program be owned by the same profile that owns the file and set the user profile attribute on the program to *OWNER.

If your program includes dynamic SQL, you must set the dynamic use profile attribute of the program to *OWNER when you compile the program if you want the dynamic SQL statement to adopt.

- ▶ Grant authority

In the case where you must update or read a file that is outside of your application's direct control or is not owned by the profile that is accessing the file, you will either must grant the profile running your application direct authority to the database file or call a program that adopts the file's owner.

- ▶ Stored procedures

For ODBC or JDBC connections where the user who is making the connection is the application user (that is, the application is accessing the data through their own IBM i profile instead of an application profile), instead of granting direct (private) authority for all application users to the files, you can call a stored procedure that is owned by a profile that either owns or has been granted authority to the files that are being accessed.

Note: Consider whether you need to secure this program from being generally called. You might want to set *PUBLIC to *EXCLUDE and grant the application users' group *USE authority.

Note: Granting application users direct (private) authority to database files is never the preferable approach. This method hard to maintain and it provides authority for application users to access the data. This access is not only through your application, but through all access methods: FTP, ODBC, JDBC, DDM, and command line.

16.3.13 Other *PUBLIC authority settings

The focus of this chapter is data, but the public authority of other application objects also needs to be considered. Here are recommendations for the appropriate *PUBLIC authority setting of other object types:

- ▶ The following types can generally be set to *PUBLIC *USE:
 - Libraries where the majority of users are running the application
 - Programs and service programs
 - Display files
 - Message files
- ▶ The following types might need to be *PUBLIC *CHANGE:
 - OUTQs and MSGQs
 - DEVDS

16.3.14 Using authorization lists to secure application objects

When working with application data, it is rare that only your application accesses that data. Over time, another application might be written that needs to read from or write to the database file. If you have secured the application database files with an authorization list, granting access to another application becomes easy. Simply grant access to the authorization list and access is immediate. If you attempt to grant private authority directly to the files, you must wait until the file is not in use. In today's 24x7 world, that might not occur until a planned outage. Authorization lists provide great flexibility to grant access at a moment's notice.

16.3.15 Integrated file system considerations

You cannot ignore data in the integrated file system (IFS). Consider the following scenarios:

- ▶ Application objects

Applications that provide a graphical user interface store the HTML and graphics in the IFS. The appropriate setting on these objects is to set them to read-only so they cannot be changed or replaced except by the administrator. This translates into setting directories and their objects to DTAAUT (*RX) and OBJAUT (*NONE). Do not forget to set the object authorities.
- ▶ Application data

Applications that use the IFS to store documents or scanned images must make sure that these objects are secure because they might contain private information. Directories should be owned by the application profile and the *PUBLIC authority should be set to DTAAUT (*EXCLUDE) and OBJAUT (*NONE). The IFS ignores adopted authority, so this means that the profile that is running the application must have sufficient authority to the application directories. To have sufficient authority, most applications that provide this function run as the application profile instead of the application user. Access to the documents and images are typically made through a client or browser interface that performs a profile swap (or a setuid/setgid) to run as the application profile. Running as the application profile provides authority to access the documents and images. Which information users are allowed to access is controlled within the application, typically through a role-based access method.

- ▶ Data that is created by an application process

Most organizations have at least one directory that contains “transient” data, which is data that is created through an application process. This process, often a batch process, typically creates a stream file that goes into a directory and that stream file is then transferred off the system with FTP or downloaded to someone's PC. Examples of these processes include payroll processing, monthly credit card charges, bank transfers, and state or federal tax processing. Because this type of process almost always contains highly confidential data, the directory must be set to *PUBLIC DTAAUT(*EXCLUDE) OBJAUT(*NONE) and owned by the profile that is running the process that generates the stream file. If any additional users need authority, only users who need to download, FTP, or check on the stream file that was generated are allowed. Do not worry about the authority of the stream file that is being generated; it is deleted and re-created every time that the process runs. If the directory is secured properly, authority to the stream file is rarely a concern.

16.3.16 Conclusion

All of these considerations for security should be made when the application is being designed. Retrofitting security features into an application that is designed and implemented is harder than integrating your organization's security requirements from the initial design phase. Unfortunately, this situation happens often. Security also is sometimes abandoned in the final test phases of an application. A feature does not work, security is blamed, and suddenly data is left with wide-open settings. An assurance is made that the data will be secured when the application is in production, but often it never happens.

Security should never be ignored. We hope this chapter helps you be more aware of security and helps you implement your organization's security requirements.

For more information, see the following resources:

- ▶ The *IBM i Security Reference* manual provides guidance for how much authority is required to perform actions on a database file and IFS directory. You can find it at the following website:
<http://pic.dhe.ibm.com/infocenter/iseriess/v7r1m0/topic/rzar1/sc415302.pdf>
- ▶ Detailed information about implementing and managing IBM i security is available in the book *IBM i Security Administration and Compliance*, available from MCPressOnline, at the following website:
<http://www.mc-store.com/IBM-i-Security-Administration-Compliance/dp/1583473734>



Globalization

Globalization is the process of international integration. This chapter looks at what globalization is about, what the requirements are to globalize your IBM i application, and an initial approach to some methodology.

17.1 Terminology

Here are some terms that often cause much confusion about characters and their conversion. Here are four of the most important terms:

- ▶ Encoding scheme
- ▶ Coded character set identifier (CCSID)
- ▶ Character set (CS)
- ▶ Code page (CP)

In Figure 17-1, you can see the structure of the different entities. Each encoding scheme can have many CCSIDs. Each CCSID uniquely identifies a character set and a code page.

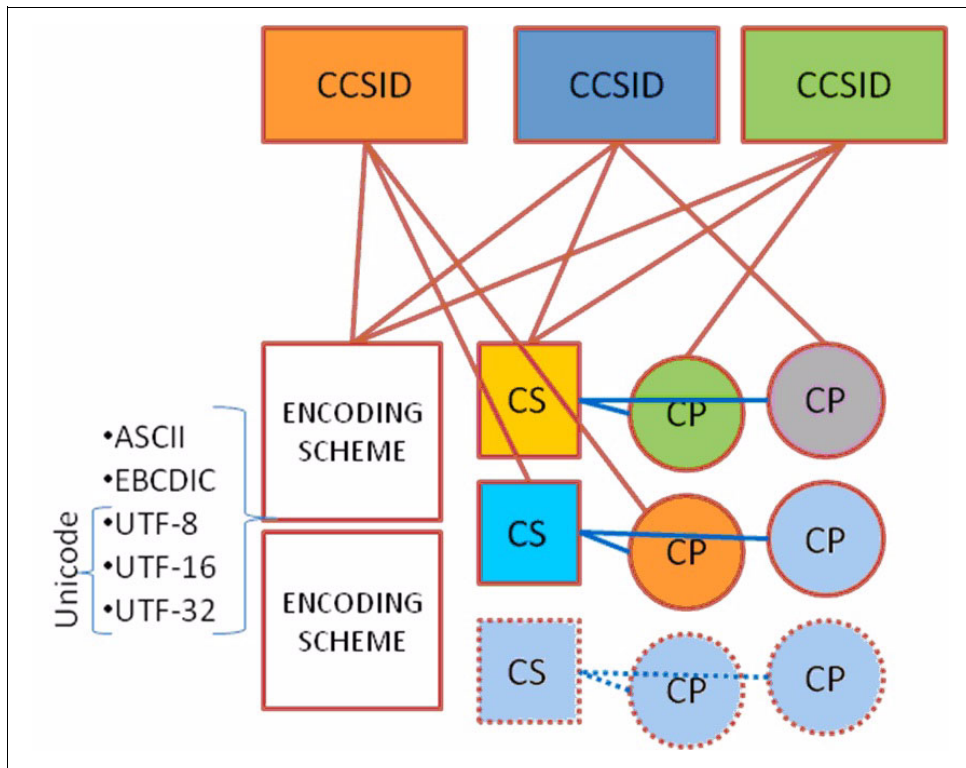


Figure 17-1 Each encoding scheme is composed of a character set and code page

17.1.1 From an encoding scheme to a CCSID

The encoding scheme is the methodology that is used to store data (encode it into the system) to represent characters.

For example, an “A” graphical representation (from a character set) that has the value 65 (decimal value in a code page) can be represented as x'41', x'0041', or x'4100', depending on the encoding scheme that is used by the system.

Here are the following primary encoding schemes:

- ▶ American Standard Code for Information Interchange (ASCII)

The ASCII encoding scheme is a character-encoding scheme that was originally based on the English alphabet. It encodes 128 specified characters, numbers 0 - 9, letters a - z and A - Z, some basic punctuation symbols, and some control codes that originate from the Teletype machine. Lastly, it includes a blank space. All of this must fit into a 7-bit binary integer.

- ▶ Extended Binary Coded Decimal Interchange Code (EBCDIC)

The EBCDIC encoding scheme is an 8-bit character encoding that is used mainly on IBM mainframe and IBM midrange computer operating systems. EBCDIC descended from the code that was used with punched cards and the corresponding six-bit binary-coded decimal code that was used with most IBM computer peripheral devices in the late 1950s and early 1960s.

- ▶ Universal Character Set (UCS) Transformation Format 8, 16, and 32 bit (UTF-8, UTF-16, and UTF-32)

UTF is a variable-width encoding scheme that can represent every character in the Unicode character set. UTF is an extendable scheme.

Each encoding scheme can have many CCSIDs that represent a wide range of *character sets* and *code pages*.

The CCSID is the important value. It is the identifier. The CCSID is a representation that defines the character set and the code page along with the encoding scheme. The series of figures in the following sections help break down a CCSID into its various parts.

17.1.2 Character set

The character set is the set of common graphical representations that is used by a culture or language to communicate by writing. It is question of graphical representation: what a human uses without any computer to write with a pen on a paper. The character set is a group of characters that can be expected to be used. This includes numbers, letters, symbols, and any language-specific characters.

Figure 17-2 shows a sample character set.

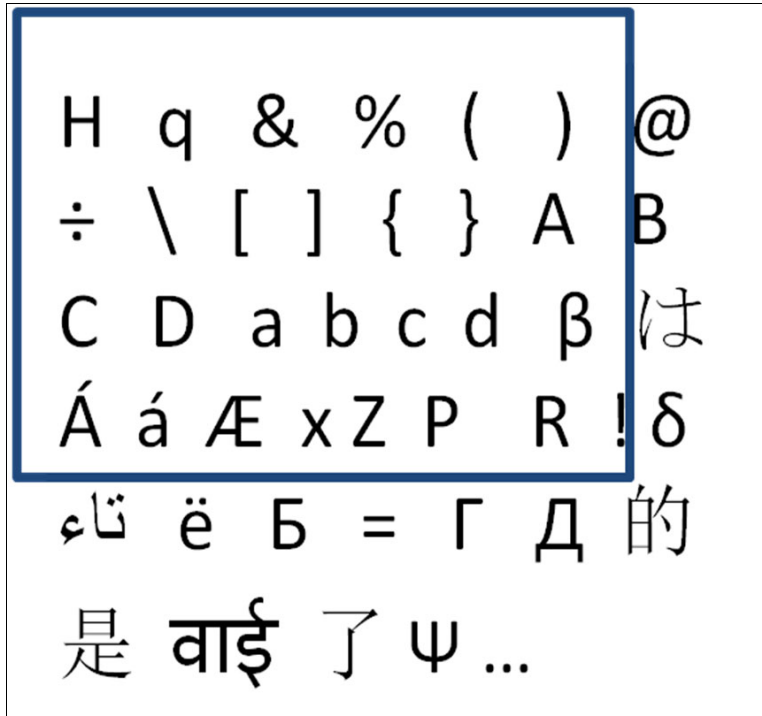


Figure 17-2 A character set is a group of characters (for example, all Western European numbers, letters, and symbols)

17.1.3 Code page

The code page belongs to a character set. It is a table that gives a unique value ID for each of the characters in the character set to which it belongs. A character set can have one or many code pages. To better illustrate this concept, Figure 17-3 shows the characters and the values for each character for code page 280. Figure 17-4 on page 648 is the code page for 00037. Both of these contain similar characters, but some of the characters can be found in different places within the table. They have a different representation value.

HEX DIGITS 1ST → 2ND ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	(SP) SP010000	& SM030000	- SP100000	ø LO610000	Ø LO620000	[SM050000	μ SM170000	¢ SC040000	à LA130000	è LC410000	ç SA060000	0 ND100000
-1	(RSP) SP300000] SM080000	/ SP120000	É LE120000	á LA010000	ĵ LJ010000	ì LI130000	# SM010000	À LA020000	Ĵ LJ020000	÷ SA060000	1 ND010000
-2	â LA150000	ê LE150000	Â LA160000	Ê LE160000	b LB010000	k LK010000	s LS010000	¥ SC050000	B LB020000	K LK020000	S LS020000	2 ND020000
-3	ã LA170000	ë LE170000	Ä LA180000	Ë LE180000	c LC010000	l LL010000	t LT010000	· SD630000	C LC020000	L LL020000	T LT020000	3 ND030000
-4	{ SM110000	ġ SM170000	À LA140000	È LE140000	d LD010000	m LM010000	u LU010000	© SM520000	D LD020000	M LM020000	U LU020000	4 ND040000
-5	á LA110000	í LI110000	Á LA120000	Í LI120000	e LE010000	n LN010000	v LV010000	@ M050000	E LE020000	N LN020000	V LV020000	5 ND050000
-6	ã LA190000	î LI150000	Ã LA200000	Ï LI160000	ŕ LF010000	o LO010000	w LW010000	¶ SM250000	F LF020000	O LO020000	W LW020000	6 ND060000
-7	â LA270000	ï LI170000	Ä LA280000	Ï LI180000	g LG010000	p LP010000	x LX010000	¼ NF040000	G LG020000	P LP020000	X LX020000	7 ND070000
-8	\ SM070000	~ SD190000	Ç LC420000	Ĩ LI140000	h LH010000	q LQ010000	y LY010000	½ NF010000	H LH020000	Q LQ020000	Y LY020000	8 ND080000
-9	ñ LN190000	β LS610000	Ñ LN200000	ù LU130000	i LI010000	r LR010000	z LZ010000	¾ NF050000	I LI020000	R LR020000	Z LZ020000	9 ND090000
-A	° SM190000	é LE110000	ò LO130000	: SP130000	« SP170000	g SM210000	ı SP030000	⌋ SM660000	̄ (SHY) SP320000	ı ND011000	2 ND021000	3 ND031000
-B	· SP110000	\$ SC030000	, SP080000	£ SC020000	» SP180000	º SM200000	ı SP160000	SM130000	ô LO150000	û LU150000	Û LO160000	Û LU160000
-C	< SA030000	* SM040000	% SM020000	§ M240000	ð LD630000	æ LA510000	Đ LD620000	ˉ SM150000	ö LO170000	ü LU170000	Û LO180000	Û LU180000
-D	(SP060000) SP070000	— SP090000	' SP050000	ý LY110000	ÿ SD410000	Ý LY120000	ˆ SD170000	ı SM650000	ı SD130000	Ò LO140000	Û LU140000
-E	+ SA010000	; SP140000	> SA050000	= SA040000	þ LT630000	Æ LA520000	Ɔ LT640000	' SD110000	ó LO110000	ú LU110000	Ó LO120000	Û LU120000
-F	! SP020000	^ SD150000	? SP150000	" SP040000	± SA020000	⌘ SC010000	® SM530000	× SA070000	õ LO190000	ÿ LY170000	Õ LO200000	€ SC0

Code Page 00280 ←

Figure 17-3 Code page 00280

HEX DIGITS 1ST → 2ND ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	(SP) SP010000	& SM030000	- SP100000	ø LO610000	Ø LO620000	° SM190000	μ SM170000	^ SD150000	{ SM110000	}	\ SM070000	0 ND100000
-1	(RSP) SP300000	é LE110000	/ SP120000	É LE120000	a LA010000	j LF010000	~ SD190000	£ SC020000	A LA020000	J LJ020000	÷ SA060000	1 ND610000
-2	â LA150000	ê LE150000	Â LA160000	Ê LE160000	b LB010000	k LK010000	s LS010000	¥ SC050000	B LB020000	K LK020000	S LS020000	2 ND620000
-3	ä LA170000	ë LE170000	Ä LA180000	Ë LE180000	c LC010000	l LL010000	t LT010000	· SD630000	C LC020000	L LL020000	T LT020000	3 ND630000
-4	â LA130000	ë LE130000	À LA140000	È LE140000	d LD010000	m LM010000	u LU010000	© SM520000	D LD020000	M LM020000	U LU020000	4 ND640000
-5	á LA110000	í LI110000	Á LA120000	Í LI120000	e LE010000	n LN010000	v LV010000	§ SM240000	E LE020000	N LN020000	V LV020000	5 ND650000
-6	ã LA150000	î LI150000	Ã LA200000	Ï LI160000	f LF010000	o LO010000	w LW010000	¶ SM250000	F LF020000	O LO020000	W LW020000	6 ND660000
-7	ä LA270000	ï LI170000	Ä LA280000	Ï LI180000	g LG010000	p LP010000	x LX010000	¼ NF040000	G LG020000	P LP020000	X LX020000	7 ND670000
-8	ç LC410000	ì LI130000	Ç LC420000	Ì LI140000	h LH010000	q LQ010000	y LY010000	½ NF010000	H LH020000	Q LQ020000	Y LY020000	8 ND680000
-9	ñ LN190000	β LS610000	Ñ LN200000	´ SD130000	i LI010000	r LR010000	z LZ010000	¾ NF060000	I LI020000	R LR020000	Z LZ020000	9 ND690000
-A	€ SC040000	! SP020000	¡ SM650000	∴ SP130000	« SP170000	ª SM210000	ï SP030000	[SM060000	̄ SP320000	1 ND011000	2 ND021000	3 ND031000
-B	· SP110000	\$ SC030000	, SP080000	# SM010000	» SP180000	º SM200000	¿ SP160000]	ô LO150000	û LU150000	Ô LO160000	Û LU160000
-C	< SA030000	* SM040000	% SM020000	@ SM050000	ð LD630000	æ LA510000	Ð LD620000	- SM150000	ö LO170000	ü LU170000	Ö LO180000	Ü LU180000
-D	(SP060000) SP070000	_ SP090000	' SP050000	ý LY110000	¸ SD410000	Ý LY120000	¨ SD170000	ò LO130000	ù LU130000	Ò LO140000	Ù LU140000
-E	+ SA010000	; SP140000	> SA050000	= SA040000	þ LT630000	Æ LA520000	Þ LT640000	' SD110000	ó LO110000	ú LU110000	Ó LO120000	Ú LU120000
-F	 SM130000	¬ SM660000	? SP150000	" SP040000	± SA020000	∩ SC010000	® SM530000	× SA070000	õ LO190000	ÿ LY170000	Õ LO200000	Ë SC0

Code Page 00037 

Figure 17-4 Code page 00037

Different code pages can be used to define different character sets. For example, the character “@” is coded x'7C' in the code page 37 or the code page 280, but it is x'44' in code page 297.

As another example, consider a book that represents a character set. Each page of the book has a table that shows the character set with different position for letters into the table. In Figure 17-3 on page 647 and Figure 17-4, you can see that you have the same set of characters, but at different position into the table.

It is possible to convert a code page to another code page if they belong to the same character set.

17.1.4 Coded character set identifier

The CCSID is a 16-bit number that describes a unique combination of a character set and a code page. It is intended to simplify the combination of the encoding schema, character set, and code page in to a single value. There is a unique CCSID for every language and schema combination. For example, in the EBCDIC encoding scheme, the CCSID for English-USA is 37, but for Italian it is 280. Normally, the CCSID number is the same as the code page, but not always. Figure 17-5 shows a brief subset of the list of CCSIDs. There are thousands of CCSIDs that are registered to cover many different languages and cultures.

Coded character set identifiers		
CCSID (decimal)	CCSID (hex)	Name
37	0025	COM EUROPE EBCDIC
256	0100	NETHERLAND EBCDIC
259	0103	SYMBOLS SET 7
273	0111	AUS/GERM EBCDIC
274	0112	BELGIUM EBCDIC
275	0113	BRAZIL EBCDIC
277	0115	DEN/NORWAY EBCDIC
278	0116	FIN/SWEDEN EBCDIC
280	0118	ITALIAN EBCDIC

Figure 17-5 Example of a small subset of CCSIDs

There are thousands of registered CCSIDs. For a complete list, see the following website:

http://www.ibm.com/software/globalization/ccsid/ccsid_registered.html

Figure 17-6 is a representation of many CCSIDs.

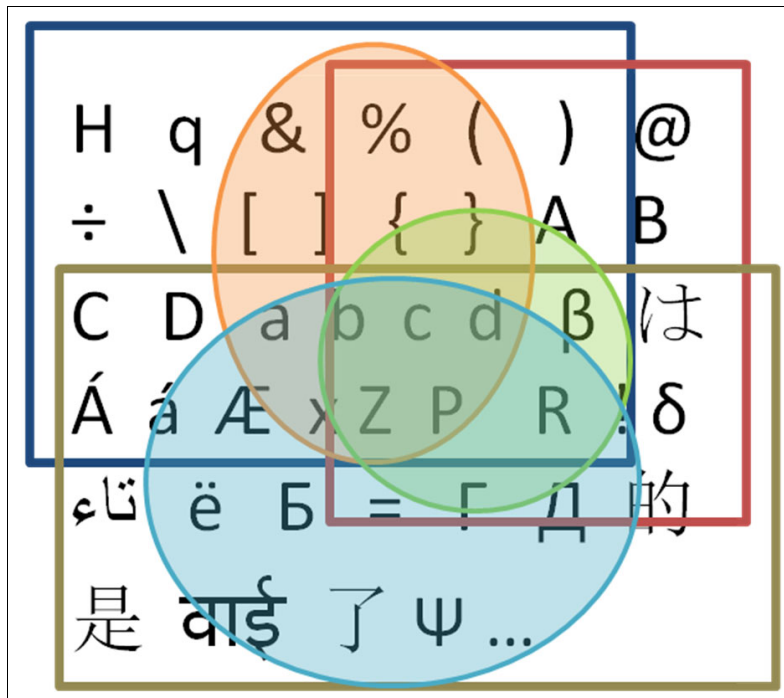


Figure 17-6 There are many character sets and code pages and therefore many CCSIDs

17.1.5 Invariant character set

Within every character set, there is a subset of characters that is common across most all character sets and code pages. This common set is *invariant*. Figure 17-7 represents the common characters.

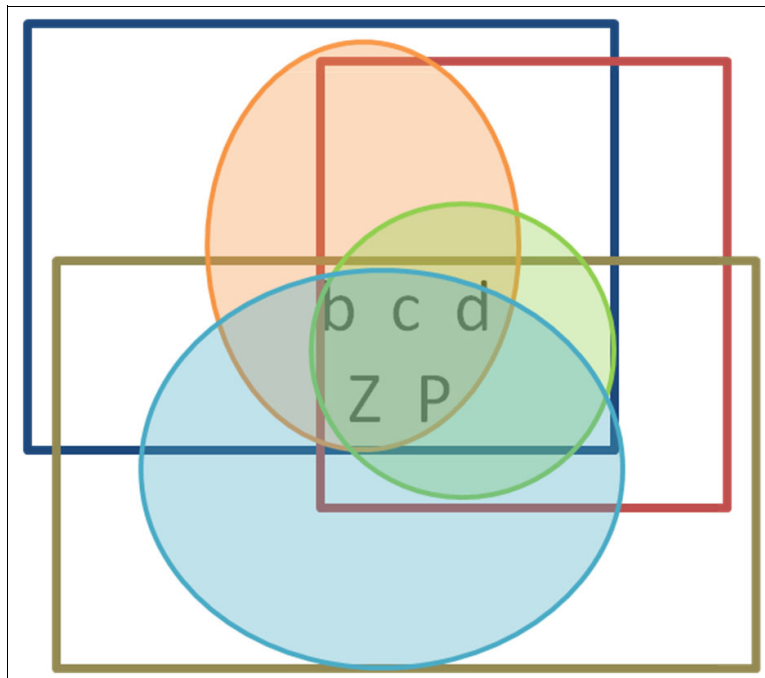


Figure 17-7 Invariant characters

In the EBCDIC encoding scheme, here are the invariant set of characters:

- ▶ ABCDEFGHIJKLMNOPQRSTUVWXYZ
- ▶ abcdefghijklmnopqrstuvwxyz
- ▶ 0123456789
- ▶ +<=>%&*"'(),_-./:;?

It is a preferred practice to use only invariant characters when you develop an application for all your objects, fields, or variable names. Otherwise, you encounter potential issues when your code runs on another IBM i system that uses a different CCSID. You should avoid naming your objects or variables with character such as \$, @, or # because they are not defined within the invariant set. For more information, see 17.3, “Data integrity between environments or languages” on page 652.

17.1.6 Unicode

Unicode is described in 17.4, “Unicode” on page 656, but the goal of Unicode is to fix the issues you might notice regarding character sets, code pages, and the many combinations. You might notice that a character set is limited in the number of characters that can be represented. Unicode handles the graphical representation for all characters for every language and culture within a single methodology. The good news is that Unicode is extendable. As new characters are invented, they can be added without having to change CCSID numbers.

17.2 Single-, double-, and multi-byte character set (SBCS, DBCS, and MBCS)

SBCS is used to refer to character encoding that uses exactly one byte for each graphic character. SBCS character sets accommodate a maximum of 256 symbols, and were originally built for English- or Western-based languages. They do not have many symbols or accented letters.

The term SBCS is commonly contrasted against the terms DBCS and MBCS. The MBCSs are used to accommodate languages with more characters and symbols than most Western languages. Multi-byte character sets are predominantly used for Eastern or Asian languages, such as Chinese, Japanese, and Korean.

In computing systems, SBCSs are traditionally associated with half-width characters, so called because such SBCS characters traditionally occupied half the width of a DBCS character on a fixed-width computer terminal or text panel.

A CCSID can use an SBCS encoding, a DBCS encoding, or also an MBCS encoding. An MBCS is a mix of SBCS and DBCS. The main reason for the mix is to save space by using SBCS whenever it is possible. The differentiation between both encodings is made with two special control characters:

- ▶ Shift-out (SO)
- ▶ Shift-in (SI)

SO means that a DBCS encoding is starting and continues until the SI character is encountered. The SI means that the DBCS characters are ending. Then, double-byte data is bracketed with SO-SI.

In hexadecimal (EBCDIC), the value of SO is x'0E' and for SI it is x'0F'. In decimal (ASCII), the values are 14 and 15.

17.2.1 DBCS and MBCS data type

Table 17-1 shows all data types that support the DBCS encoding.

Table 17-1 DBCS data type

Field type	Definition	Description
O	DBCS-open	A character string that contains both single-byte and bracketed double-byte data.
E	DBCS-either	A character string that contains either all single-byte data or all bracketed double-byte data.
J	DBCS-only	A character string that contains only bracketed double-byte data.
G	DBCS-graphic	A character string that contains only non-bracketed double-byte data.

Most of the time, types O and G are used.

For a long time, DBCS was used in 5250 displays and printers because Unicode was poorly supported by most emulators until recently.

17.3 Data integrity between environments or languages

When you must exchange data between *environments* or *languages*, you must consider data integrity. Data integrity is when every character that you consider can be transposed from one environment to another. This round trip of data conversion is when all characters exist in both character set environments.

Data integrity in the IBM i is activated by CCSID. When the source and destination data have different CCSIDs, the system tries to convert characters to keep their representation.

17.3.1 Environments

On the IBM i, there are many different environments that must be considered. Additionally, you must consider the interaction with other platforms:

- ▶ The IBM i OS is fundamentally an EBCDIC system.
- ▶ The IBM PASE for i is fundamentally an ASCII system.
- ▶ The IBM i IFS is fundamentally an ASCII system.
- ▶ Other platforms than the IBM i can be EBCDIC or ASCII.

17.3.2 Languages

There are 60 languages that are supported on the IBM i system. The default language for the systems is set by the system value Language ID (QLANGID). You can also prompt and set a different language ID for any user profile on the system by setting the user ID attribute, language ID LANGID.

17.3.3 Why a CCSID is important

The CCSID can have a value 0 - 65535. Any entity such as an object, source file, or a job has a CCSID. The CCSID indicates which code page and character set are used for processing. When an entity processes with a different CCSID, the system operates a conversion mapping.

For example, for the country US, the CCSID is 37 and the character “\$” is represented with the hexadecimal value x'5B'. If this character must be sent to a UK job, its hexadecimal value is converted to x'4A', which is the representation of “\$” in the UK CCSID 285. This is how data integrity is safeguarded.

Nevertheless, there is a special CCSID value that turns off automatic conversion. This value is 65535. This value is not recommended and it is a preferred practice to set your system value QCCSID to an effective default CCSID.

The reason for this setting is simple. If conversion must be done, you must know what the source and target character and code page set are so that they operate properly. In some situations, such as transferring data to another encoding scheme, the system has no choice but to try to guess what the CCSID is if it is not indicated (if it is equal to 65535). The clue the system then uses is the language ID, from which a default CCSID can be determined.

Note: The overall rule is that data integrity is maintained by using a CCSID value.

You can lose characters if they do not exist on the target CCSID's character set. In this case, you can take the following actions:

- ▶ Replace characters that do not exist by using the substitution character x'3F' (EBCDIC x'3F' is represented with the question mark (?) character in ASCII).
- ▶ Replace characters that do not exist by using a character that is “close” (for example, replace 'ä' with 'a').
- ▶ Replace characters that do not exist by using a character that does not exist in the source CCSID (for example, replace 'ä' with '?'). The advantage is, for a round trip, the '?' can be converted back to the 'ä'.

These manipulations might necessary if your source CCSID contains a character that does not exist in your target CCSID (remember, a CCSID is made of a character set and a code page). These manipulations can be avoided if you use a Unicode CCSID (Unicode is described in 17.4, “Unicode” on page 656).

17.3.4 Does a program have a CCSID

Yes, programs have a CCSID, and this is relevant for internal names, constants, and literals that use non-variant characters.

When is this potentially relevant?

- ▶ When you transfer source code to another system with a different CCSID environment.
- ▶ When your program compares job character values to program literals in another CCSID environment.

For ILE programs, the CCSID of the compiled program is the CCSID of the primary source file. The CCSID of the program or service program are often set to 65535. So, you must watch the module's CCSID.

For non-ILE programs, the source from the database source files is converted to the CCSID of the job. Therefore, the CCSID of the compiled program is the CCSID of the job on which the program is compiled. If you do not want your names, constants, or literals to be converted to the CCSID of the job, you can change your job CCSID to 65535.

Compilers “speak” only binary for the code. For example, in RPG, some variant characters are accepted to define a variable name. This means that you must be careful about variable names, depending the CCSID of your source files, because the variable names can change depending on the CCSID:

- ▶ In a source file with CCSID 37 on a US Page Code panel, the “@VAT” variable name is accepted.
- ▶ If you type this same code into a 297 CCSID coded source file, you must type “àVAT” because it has the same binary value (7C).

This situation causes a problem if you copy source that is using variant characters for variable names. If you copy the source, you must manually convert each variable and constant or your program might no longer compile correctly.

Note: The preferred practice to safeguard integrity of non-variant character literals in your program is one of the following practices:

- ▶ Use an external file to contain them.
- ▶ Describe them internally in Unicode.

Both methods associate a CCSID tag to your character’s literals.

When a file has a CCSID tag, the system automatically processes any necessary conversion for the job at run time.

Example 17-1 shows you how to describe a literal internally by using Unicode in RPG.

Example 17-1 Defining the literal hash # and dollar \$ in Unicode in RPG

d hash	S	1C	Inz(%UCS2('#'))
d dollar	S	1C	Inz(%UCS2('\$'))

Does a display or printer file have a CCSID

Yes, they have a CCSID, and this is relevant for any internal names, constants, and literals that use non-variant characters.

In addition, display or printer files can use the DDS keyword CCSID for any I/O field, but your final devices (such as emulators and printers) must handle it.

17.3.5 Flow examples

In the following examples, you see a classical situation of a flow from database access to display interaction. The CCSID flow is highlighted between the different elements that are described in Figure 17-8 on page 655.

This section lists the elements to consider:

- ▶ For literals and constants:
 - Program
 - Display or printer file

Your program and display or printer file can contain constants or literals.

- ▶ For data fields and variables:
 - File
 - Program
 - Display or printer file if it contains specific CCSIDs field DDS keywords

Every time a field value is moved to another field that has a different CCSID, a conversion is processed.

- ▶ For a visualization: UI device (terminal, GUI, or printer)
 - The UI device must be configured to be able to represent all character sets.

Figure 17-8 shows the flow from the database to the UI device.

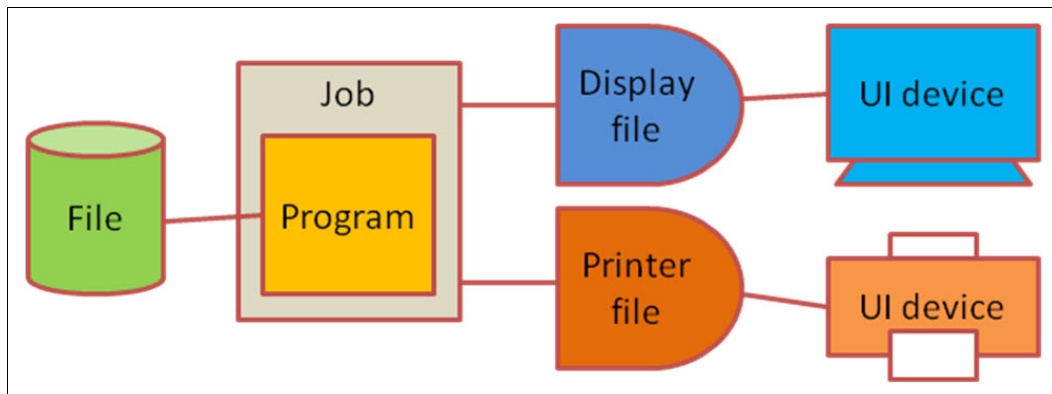


Figure 17-8 Flow from the database to the UI device

17.4 Unicode

The Unicode standard is the universal character encoding standard that is used for representation of text for computer processing (Figure 17-9). Unicode provides a consistent way of encoding multilingual plain text to make it easier to exchange text files internationally and to represent multilingual characters.

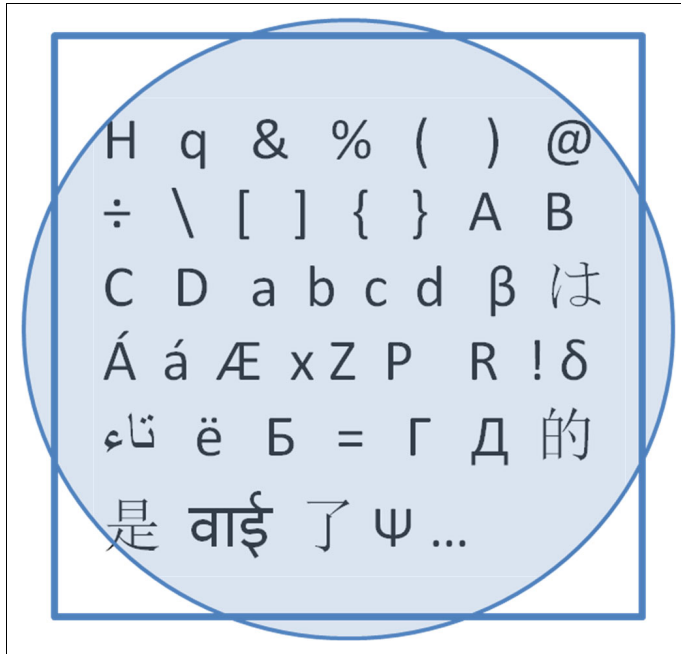


Figure 17-9 Unicode represents a global character set that includes all characters

17.4.1 Unicode CCSID

There are CCSIDs that can represent Unicode. Table 17-2 illustrates the supported CCSIDs for Unicode.

Table 17-2 Unicode CCSIDs

CCSID	Encoding	Data type	SQL	
13488	UTF-16 DBCS	G	Graphic	Fixed Length: 1 char = 2 bytes
1208	UTF-8 MBCS	A	Char	Variable Length: 1 char = 1 - 6 bytes
1200	UTF-16 DBCS	G	Graphic	Variable Length: Most of the time 1 char = 2 bytes, but in some less common languages, 1 char = 4 bytes

A data type in a Program “A” (Alphabetical) might contain the following items:

- ▶ SBCS encoding space
- ▶ Mix
- ▶ UTF8

So, you must know when you work with the data type what data it contains.

For a database field coded CCSID(1208) that contains 'aè' in SQL, the Function Length(FieldName) returns 3 (the number of bytes, and not the number of characters because UTF-8 has variable byte length for common characters).

Within IBM, UTF-8 is registered as CCSID 1208 with a growing character set (sometimes also referred to as code page 1208). As new characters are added to the standard, this number (1208) does not change.

Within IBM, UTF-16 is registered as code page 1200, with a growing character set (sometimes also referred to as code page 1200). When new characters are added to this code page, the code page number does not need to change. Code page 1200 always refers to the current version of UTF-16 Unicode.

A version of the Unicode standard, as defined by Unicode 2.0 and ISO/IEC 10646-1, is registered within IBM as CCSID 13488. On IBM i CCSID 13488 and CCSID 1200, both refer to UTF-16, and are handled the same way. The same conversion support is used for both CCSID 1200 and CCSID 13488.

The preferred CCSID for UTF-16 is 1200 because it represents the cross-platform version of UTF-16, whereas 13488 is an older CCSID and might not be supported on other platforms.

Space and MBCS conversions: One item to note about conversion between Unicode and MBCSs on the system is the mapping of the space characters. MBCS strings, such as Japanese, can contain two “space” characters. One is used in the SB part of data and one is used in the DB part. When it is converting MBCS items to Unicode, the system converts the SB space to U+0020 and the DB space to U+3000. This is unique to IBM conversions and other vendors might or might not recognize U+3000 as a “space”.

17.4.2 Unicode examples from the DB through the RPG to the DSPF

In the following examples, you see a Unicode database that is controlled by an interactive RPG program.

First, you must define the database table. Example 17-2 is the table definition using Unicode.

Example 17-2 Table with two Unicode fields

```
CREATE TABLE MYLIB/UNICODEF (  
  "CHAR" CHAR(20) CCSID 37 NOT NULL DEFAULT '' ,  
  CHAR2 CHAR(20) CCSID 37 NOT NULL DEFAULT '' ,  
  UNICODE GRAPHIC(20) CCSID 1200 NOT NULL DEFAULT '' ,  
  UNICODE2 GRAPHIC(20) CCSID 1200 NOT NULL DEFAULT '' ,  
  NUMBER DECIMAL(5, 0) NOT NULL DEFAULT 0 ,  
  NUMBER2 DECIMAL(5, 0) NOT NULL DEFAULT 0 )  
  
RCDFMT UNICODR ;
```

Example 17-3 shows the definition of the table in a DDS file.

Example 17-3 Table in DDS - physical file

A	R UNICODR		
A	CHAR	20	
A	CHAR2	20	
A	UNICODE	20G	CCSID(1200)
A	UNICODE2	20G	CCSID(1200)

A	NUMBER	5	0
A	NUMBER2	5	0

Example 17-4 shows the RPG program that is using the Unicode database table.

Example 17-4 RPG program

```

H CCSID(*GRAPH: *SRC)
H CCSID(*UCS2: 1200)
Funicodd  CF  E                WORKSTN
Funicodf  UF  A  E            K  DISK
F                                                prefix(f_)

d xUNICODE      s                20C  ccsid(1200)
d xUNICODE2     s                20C  ccsid(1200)
d xCHAR         s                20

/free
    read unicodf;
    exsr file2screen;

    dow *in03 = '0';
        exsr DSPsr;
        exsr PRCsr;
    enddo;

    exsr screen2file;
    if %eof;
        write unicodr;
    else;
        update unicodr;
    endif;

    *inLR = *on;

/*=====
begsr DSPsr;
/*=====
    exfmt SC1;
endsr;

/*=====
begsr PRCsr;
/*=====
// In this example we will use explicit conversion
// between field of different data type and CCSID,
// in using the BiF %char() and %ucs2().

// Note that since v6 (PTF available for v5r3, v5r4)
// the RPG rules have been relaxed for the following:
// A UCS-2 field can be initialized using INZ using an
// alphanumeric or graphic literal - INZ('abc') or INZ(G ' ').
// In the following cases, the left side and right
// side can be any combination of alphanumeric, UCS-2 and
// graphic data. Implicit conversions are performed if
// necessary.
// Nevertheless free-form concatenation using the "+"

```



```

// operator requires explicit conversions and
// BiF using string operations.

// Move Unicode field to rpg-char variable,
// characters not in char-set-ccsid target will be lost:
eval xCHAR      = %char(UNICODE);

// Move rpg-char variable to Unicode field:
eval xUNICODE   = %ucs2(Char);

// Move Unicode field to rpg-Unicode variable:
eval xUNICODE2  = UNICODE;

eval CHAR2      = CHAR;
eval UNICODE2   = UNICODE;
eval NUMBER2    = NUMBER;
endsr;

begsr file2screen;
eval CHAR       = f_CHAR      ;
eval CHAR2      = f_CHAR2     ;
eval UNICODE    = f_UNICODE   ;
eval UNICODE2   = f_UNICODE2  ;
eval NUMBER     = f_NUMBER    ;
eval NUMBER2    = f_NUMBER2   ;
endsr;

begsr screen2file;
eval f_CHAR     = CHAR       ;
eval f_CHAR2    = CHAR2      ;
eval f_UNICODE  = UNICODE    ;
eval f_UNICODE2 = UNICODE2   ;
eval f_NUMBER   = NUMBER     ;
eval f_NUMBER2  = NUMBER2    ;
endsr;

```

Example 17-5 defines the display file that is used by the RPG program.

Example 17-5 Display file

```

A                                     DSPSIZ(24 80 *DS3)
A                                     PRINT
A                                     HELP
A          R SC1
A                                     CF03(03)
A                                     1 33'Unicode example'
A                                     COLOR(WHT)
A                                     2 26'Process between display/RPG/DB'
A                                     COLOR(WHT)
A                                     4 2'File fields:'
A                                     COLOR(BLU)
A                                     5 2'CHAR'
A          CHAR          20A B 5 30CHECK(LC)
A                                     6 2'CHAR2      = CHAR'
A          CHAR2        20A 0 6 30
A                                     7 2'UNICODE'

```

```

A          UNICODE          20G  B  7 30CCSID(1200)
A          CHECK(LC)
A          8 2'UNICODE2 = UNICODE'
A          UNICODE2        20G  0  8 30CCSID(1200)
A          9 2'NUMBER'
A          NUMBER          5S 0B  9 30
A          10 2'NUMBER2 = NUMBER'
A          NUMBER2        5S 00 10 30
A          12 2'RPG variables:'
A          COLOR(BLU)
A          13 2'xCHAR      = %char(UNICODE)'
A          XCHAR          20A  0 13 30
A          14 2'xUNICODE  = %ucs2(CHAR)'
A          XUNICODE        20G  0 14 30CCSID(1200)
A          15 2'xUNICODE2 = UNICODE'
A          XUNICODE2      20G  0 15 30CCSID(1200)
A          18 2'Insert values in char and unicode.'
A          COLOR(BLU)
A          19 3'(copy/paste values from web in Chi-
A          nese, Greek, Korean ...etc)'
A          COLOR(BLU)
A          20 2'Press Enter to see RPG variables c-
A          hanges and to update the file.'
A          COLOR(BLU)
A          21 3'(The file will contain only 1 reco-
A          rd at a time.)'
A          COLOR(BLU)
A          23 2'F3=Exit'
A          COLOR(BLU)

```

17.4.3 Moving to Unicode

To change your existing database and code to Unicode, here are the main principles to follow:

- ▶ For your file:

You can use either ALTER FIELD, CHGPFM, or CPYF.

- ▶ For your ILE RPG:

RPG does implicit conversions for assignment and comparisons between A, G, and C.

However, your program might not work the way that you want it to work because conversions from UCS-2 to DBCS or to single byte can lose data if there is a character in the UCS-2 data that does not have a match in the single-byte data.

The new **CCSIDCVT H** spec keyword (added with PTFs for 6.1 and 7.1) can help. If you code **CCSIDCVT(*LIST)**, you can get a section in the listing that lists every CCSID conversion in the module. Then, you can use that information to help locate the work fields that must be changed to UCS-2.

If you specify **CCSIDCVT(*EXCP)**, you get an exception at run time if a conversion causes data loss.

For more information about the **CCSIDCVT** keyword, see the following website:

<http://ibm.biz/BdxXWJ>

17.4.4 Emulators

Most of 5250 emulators have no Unicode capabilities. Only the terminal character set can be displayed. This enables only one language to be displayed on the panel at a time.

There are a few 5250 emulators that do handle Unicode. With them, all Unicode characters can be displayed on the same panel and you can have multiple languages that are displayed (such as Chinese and Russian). Here are the main Unicode capable 5250 emulators:

- ▶ iAccess for Web
<http://www.ibm.com/systems/power/software/i/access/web.html>
- ▶ Integrated HATS emulator
<http://www.ibm.com/software/products/en/rhats/>
- ▶ IBM Access Client Solutions
<http://www.ibm.com/systems/power/software/i/access/solutions.html>
- ▶ IBM Host On Demand
<http://www.ibm.com/software/products/en/rationalhostondemand/>

The following ISV products are also available:

- ▶ ARCAD 5250 Free Virtual emulator (in Eclipse)
- ▶ openlook from looksoftware

17.5 Locales

A locale is an object that can determine how data is processed, printed, and displayed. Locales are made up of categories that define language, cultural data, and character sets. These combinations of language, cultural data, and character sets make up a locale.

Table 17-3 shows a list of system values that define locale information.

Table 17-3 Example of locale system values for Japanese

Keyword	Description	Example for JAPANESE Katakana - DBCS
QCCSID	Character set identifier. This is the recommended QCCSID value if you want to use CDRA support. For all national language versions (NLVs), the default QCCSID value is 65535 unless otherwise indicated.	05026
QCHRID	Character set and code page.	01172 00290
QCNTYID	Country or region identifier.	JKB
QCURSYM	Currency symbol. The values that are given are accurate; however, the system supports only one character in that return field.	Yen sign
QDATFMT	Date format.	YMD

Keyword	Description	Example for JAPANESE Katakana - DBCS
QDATSEP	Date Separator.	Hyphen (-)
QDECFMT	Decimal Format.	Blank
QIGC	DBCS version indicator.	1
QIGCCDEFNT	DBCS font name.	QFNTDBCS/XZEFFB0
QKBDTYPE	Keyboard type.	JKB
QLANGID	Language identifier.	JPN
QLEAPADJ	Leap year adjustment.	0
QSRTSEQ	Sort sequence.	*HEX
QTIMSEP	Time separator.	Colon (:)
Internet CCSID	Client character set environment.	00942
Client encoding nomenclature	National Language Technical Center (NLTC) value and Document Encoding. Client encoding nomenclature provides a guideline for configuring a client for a specific language and setting up your internet web browser.	ShiftJIS

17.6 Reference links

The following resources contain information about globalization and Unicode:

- ▶ Globalize your business:
<http://www.ibm.com/software/globalization/>
- ▶ *e-business Globalization Solution Design Guide: Getting Started*, SG24-6851
- ▶ Unicode:
<http://unicode.org>



Cloud

Now that you have a modern application, or at least are considering the process, what happens when you want to run your application in the cloud? This chapter reviews some of the principles of what cloud is all about and some things that you must consider.

18.1 Overview: IBM i and cloud

The fact that the IBM i operating system (OS) is one of the best multi-workload, multiuser business platforms is undeniable. The value of IBM i and its architecture, which is based on object security, job descriptions, robust job scheduling, and work management, make it an ideal platform for running multiple process workloads. IBM i users run batch processing, web, and interactive application workloads concurrently, while still enjoying high availability and security. Other platforms and operating systems are not structured in such a way to run these types of workloads as efficiently. Managed service providers (MSPs) and cloud providers are embracing IBM i as a solid enterprise cloud platform that they can use to run their customer's business environments.

18.2 IBM i cloud considerations

Cloud infrastructure providers, MSPs, and SaaS application providers must consider several things when creating and supporting applications to deploy as services in a hosted environment:

- ▶ Security and isolation
- ▶ New tenant provisioning
- ▶ Service level agreements (SLAs) and monitoring
- ▶ Licensing and pricing
- ▶ Deployment options

18.2.1 Security and isolation

Security and isolation of cloud tenants from each other's business processes and data is a primary concern. Typically, the operating system and middleware contain the mechanisms to provide authorization and authentication services that allow access and the capabilities to compartmentalize the data that tenants can view. The application provider must know how to interface and use these built-in IBM i capabilities, thus avoiding the need to write them in to the application. Capabilities including IBM i subsystems and job management, workgroups, independent auxiliary storage pools (IASPs), memory pools, integrated IBM DB2 database, and isolated IBM i partitions are IBM i capabilities that provide tenant separation.

18.2.2 New tenant provisioning

New tenant provisioning is the ability to create the infrastructure, platform, and software environment in a timely and repeatable manner for your new customers. Some cloud technologists claim that the target for this is in real time and measured in minutes or hours. However, the IBM i infrastructure provider that runs a Software as a Service (SaaS) application takes more time to set up a tenant because these IBM i SaaS applications tend to be enterprise resource planning (ERP), business-critical applications. Several features exist in IBM i to make new tenant provisioning possible: The IBM i save and restore mechanisms, network storage spaces, partitioning capabilities, and processor sharing and capping are all technologies that the solution provider can employ. With IBM i V7R1 Technical Refresh level TR3 and beyond, IBM enables key technologies in the OS to allow cloud provisioning (capture, deploy, suspend, and mobility) operations for IBM i. The ability to capture for repeated, rapid deployment (within minutes) of new IBM i workloads and environments is now built in to the OS.

18.2.3 Service-level agreements

SLAs are necessary to ensure customer satisfaction. Contractually ensuring performance, uptime, and disaster recovery are focal topics that you need to address. You must engineer and design the SaaS application to run as a managed process. In a single IBM i instance, the subsystems, memory pools, and job descriptions (although used more by the hosting, infrastructure provider or data center owner) help isolate applications from impacting other tenants in the cloud infrastructure. It is also possible to run IBM i in a virtual server or logical partition (LPAR) environment, where separate IBM i OS stacks can run single tenants, isolating them from each other. With IBM Capacity on Demand, and processor and memory sharing and capping capabilities, you can assign tenant resources and dynamically tune them for service levels that pertain to system performance and response or transaction levels. There are also blended environments where there are multiple partitions each running multiple tenants that might have similar tenant requirements or characteristics, for example, geographic location, or similar application requirements or serviced industry. The hosting provider must monitor the system performance and identify problems and resolve issues. Management and monitoring tools for IBM i include Management Central and System i Navigator, iDoctor performance analysis, DBMon, and Visual Explain database monitoring, and the integrated solutions console. Virtual I/O Server (VIOS) performance advisor also allows for collection and monitoring of IBM i virtualized environments running in the cloud infrastructure.

18.2.4 Licensing and pricing

In this delivery paradigm, it is important to consider application licensing and pricing models. The classic model is that of purchasing software to be run on the premises in the customer-owned data center. This traditional model is replaced by licensing software and applications in an on-demand transaction-centric manner. It is based on the number of customer accounts that the software manages for the hosted customer, an agreed to number of transactions per billing cycle, or based on a monthly subscription. It might be necessary to modify the application to do usage metering for billing purposes. The application now must be cognizant of runtime and client usage data that must be collected for billing purposes. This usage data can be grouped into some general categories of chargeable usage units.

- ▶ Physical units that are based on infrastructure and OS resources that include the number of users, number of processors that are used, number of client accounts, or number of client connections to the SaaS application. These system level usage metrics can be collected through tools that work with IBM i OS, such as IBM Tivoli® Usage and Account Management (TUAM) and by setting up IBM i job accounting for tenant jobs and environments.
- ▶ Application-centric, where the “unit of work” or transaction is a single execution of an action, such as a single financial transaction.
- ▶ A more complex process flow unit where pricing is based on a broader set of transactions and flows that make up a single, larger business process, for example, underwriting a new insurance policy, which is a multi-step work and document flow.

18.2.5 Deployment options

Options for deploying and hosting the SaaS application fall into two broad categories.

- ▶ Self-hosting the application in the SaaS application provider's data center. Infrastructure processes, including provisioning of compute resources, high availability and disaster recovery, and performance service levels, are owned by the SaaS provider. Hardware resources are also owned by the SaaS provider.

- Outsourcing of the infrastructure piece to a data center provider or MSP. The SaaS application provider partners with a data center hosting provider who owns the infrastructure processes. The hosting provider might own the hardware resources or the SaaS provider might co-locate their hardware resources at the data center of the hosting provider. Options for data center hosting providers include IBM Services and third-party providers.

18.3 IBM i OS cloud and multitenancy

Figure 18-1 shows IBM i multitenancy models.

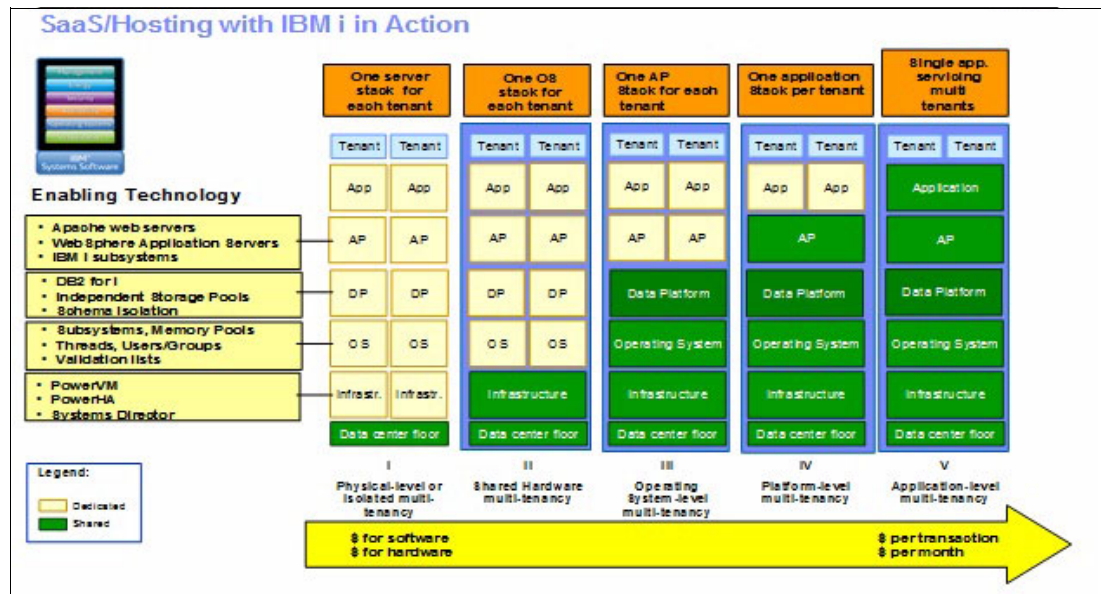


Figure 18-1 IBM i multitenancy models

Here are some definitions for the blocks that are shown in Figure 18-1 as they relate to the IBM i operating system:

- **Data center floor**
The physical location of the hardware.
- **Infrastructure**
The layer above the data center floor layer is the infrastructure layer. It includes the hardware resources and management and monitoring functions that are required to host the software application. The products for IBM i that provide these capabilities are IBM PowerVM®, IBM PowerHA, VIOS, and LPAR functions, and the evolving IBM System Director and IBM Tivoli capabilities that allow for provisioning and creating OS images and partitions and monitoring image and OS health and status.
- **Operating system**
The next block is the OS itself, which provides job scheduling and processor management, memory management, and multi-user capabilities. IBM i provides the work management infrastructures that include subsystems, memory pools, job/process/thread management, system directory, and validation lists for user and group security, and Independent Auxiliary Storage Pools. All of these can be used to provide containers and isolation capabilities for SaaS applications.

- ▶ Data platform

The next layer is the data platform (DP) layer. This is the domain of DB2 for IBM i. DB2 for IBM i provides all the relational database support and access for applications.

- ▶ Application platform

Above the DP layer is the application platform (AP) layer, which is the runtime containers where an application is deployed and run, providing application isolation and runtime services. Java Platform, Enterprise Edition application servers, like WebSphere and Web Logic or .NET application servers, can be viewed as application platforms. On IBM i, this can be the IBM WebSphere Java Platform, Enterprise Edition server, but it can also be subsystems running IBM i programs and jobs that are written in RPG using the work management capabilities of IBM i, or the Apache HTTP web container running common gateway interface (CGI) or PHP web applications.

- ▶ SaaS business application

The topmost layer is the SaaS business application itself.

18.3.1 Physical/isolated multitenancy

The first model (physical/isolated multitenancy in Figure 18-1 on page 666) is an option for SaaS and hosting, but it is not “multitenancy” at all. As its name implies, any new customer or tenant is serviced by a new hardware footprint that runs the application isolated from other tenants running on their own, separate hardware and OS stacks. The separate hardware footprint might or might not be owned by the tenant. The term *collocation* is used for this model where the hardware is owned by the customer but is in and is managed in the MSP infrastructure for their use.

18.3.2 Shared hardware multitenancy

The second model is the shared hardware multitenancy model. Figure 18-1 on page 666 shows a shared hardware infrastructure with shared resources that include network adapters, memory, disk, and processors. With the capabilities of PowerVM, you can partition these resources to create and run each tenant SaaS application in a separate stack from the OS layer and up. Each tenant runs in its own IBM i partition with an isolated and separate database and application server that is dedicated to that tenant.

18.3.3 Operating system level multitenancy

In the operating system level multitenancy model, multiple tenants run in and share the capabilities of one OS instance. In this model, however, each tenant application runs in a separate application platform or subsystem. On IBM i, multiple instances of the application server can be created to run the tenant application isolated from other application server instances running their tenant's application. Alternatively, an IBM i subsystem can be created for each tenant, each running the specific tenant application. Application isolation is attained through separation at the application platform layer and sometimes at the data platform layer.

There are different options for isolation at the data platform layer. Before the capabilities of Independent Auxiliary Storage Pools (IASPs), an instance of IBM i allowed only one database instance to be configured, unlike DB2 for Linux, UNIX, and Windows, where multiple databases can be created and managed. With the advent of IASPs in IBM i, there are two strategies to isolate the tenant's data:

- ▶ The application can isolate different tenants by intelligently separating tenant data into separate schemas (libraries). Using user group properties, job descriptions, and library list management, tenants can access only their own schemas and data.
- ▶ IASP support allows for DASD resources to be separated for each tenant. Each IASP has its own database instance and job descriptions for users and groups can point to the correct IASP and data repository.

18.3.4 Platform-level multitenancy

The platform-level multitenancy model proposes that there is an instance of the SaaS application for each tenant, but all applications are in and run in one shared server or main process. WebSphere, for example, can install an enterprise application for each tenant and has the infrastructure that is built in to handle isolation and security. Traditional IBM i subsystems can behave like an application server container and start one job per tenant, creating the secure isolation. As in the operating system level multi-tenancy model, the data platform isolation can be done in a similar manner through schema and library list isolation.

18.3.5 Application-level multitenancy

The application-level multitenancy model specifies that multiple tenants are running in the same application process and the same application platform and data platform. For example, WebSphere Application Server is threaded and creates a thread of execution for the new request running in the shared application container. The same thing can be done with traditional IBM i program jobs that have a main server process that creates job processes or implements a threading model. These threads are really the same generic code for each user. There is no separate, customized application that is specific to a tenant. This single application is responsible for identifying and isolating the tenants from each other. This application controls which business data each tenant is allowed to access in the same database. On IBM i, the data platform isolation can be done more easily by using schema and library list isolation, and on other data platforms and databases, the application is responsible for data separation.

There are many more variations of these models, including the multitier models that separate the data platform, the application platform, and the application on separate hardware, partition, or OS stacks. For example, DB2 for i can catalog databases on other IBM DB2 servers. In this case, the database interaction is to a remote data platform. The SaaS application can provide data isolation in this manner by having remote data platform and database instances per each tenant.

IBM i is a robust business platform that has underpinnings in its architecture that allow application providers to build applications that can be deployed as a SaaS application in a cloud and MSP infrastructure.

18.4 IBM i cloud technical refresh capabilities

IBM i cloud implementation and tool strategy focuses on the shared hardware multitenancy model that is shown in Figure 18-1 on page 666. The physical infrastructure is shared between several operating system instances or tenant environments. There is a hypervisor that owns the resources and controls allocation to the separate virtual server instances. The ability to implement cloud environments on Power Systems has a prerequisite that the infrastructure is virtualized using SAN storage and VIOS. IBM i MSPs and ISVs must move from Power Systems hardware with internal disks to virtualized environments with SAN storage infrastructure. VIOS is run as the “hosting” operating system that owns all the infrastructure resources, including processors, memory, storage, and networking. Tenants are deployed to new virtual server instances of IBM i, AIX, or Power Systems running Linux running as guests of VIOS.

18.4.1 Virtualization

Virtualization is the concept of running software in an environment that minimizes or eliminates the software's dependence on the hardware on which it runs. This is a critical IT concept that can be used to satisfy today's demand for managing dynamic workloads, optimizing computing resources, and quickly responding to requests for additional computing capacity. By using this technology, you can virtualize one or more of the resources that are used to build your IT infrastructure, resulting in a wide range of benefits for both the provider and consumers. Virtualization of resources is a prerequisite for building a successful cloud infrastructure, and the effective management of a virtualized environment has become critical in the pursuit of maximum benefits.

The IBM i operating system (formerly known as i5/OS) has always been known for its innovative technology to run multiple applications in a single environment, virtualize storage with its single-level storage, and to integrate security, DB2 for i, and other middleware. In addition, IBM i network installation and virtual media support provide unbeatable “hands-off” installation and save capabilities. The combination of IBM i and IBM PowerVM deliver powerful virtualization technologies and together with IBM Power Systems offer an integrated solution for managing unpredictable growth.

When building an infrastructure of hardware resources, typically the more “virtualized” the resources, the more this pool of hardware resources can be used, and the greater the benefits. Without some virtualized resources, you cannot expect to see the benefits of cloud computing. The more virtualized the infrastructure you are using to serve your clients, the more benefits are seen by both you and your consumers.

18.4.2 Virtualization technologies for cloud and IBM i

Starting in IBM i V7R1 Technical Refresh 3 (TR3), IBM has added key technologies to the IBM i OS:

- ▶ PowerVM Hypervisor IBM i support:
 - Virtual machine and server environments with shared pools of processor and memory
 - Can provide virtual I/O resources to the IBM i, AIX, and Linux on Power operating systems
- ▶ Virtualized storage and networking with VIOS and external storage IBM i Support: Basis for satisfying many virtualization requirements on IBM PureSystems®
- ▶ Network installation: Provides dynamic installation capability for software products

- ▶ Suspend/Resume
 - Can suspend a workload
 - Minimizes resource usage for environments that have varying usage requirements
- ▶ IBM Active Memory™ Sharing: Intelligently exchange memory between running partitions and virtual servers
- ▶ Provisioning: Create a virtual server quickly and without errors by using IBM System Director V6.3 and VMControl 2.4
- ▶ Partition Mobility:
 - Move a running partition to another system
 - Provides elasticity and availability (and nondisruptive maintenance)
- ▶ SmartCloud Entry: Cloud automation and self-managing technology
- ▶ IBM Image Construction Toolkit (ICCT): Tools to create and assemble cloud deployable images that include IBM i OS

18.5 IBM i virtual appliances

The ability to create images that can be deployed to an MSP/cloud environment is provided through ICCT. Figure 18-2 shows how IBM i images are constructed.

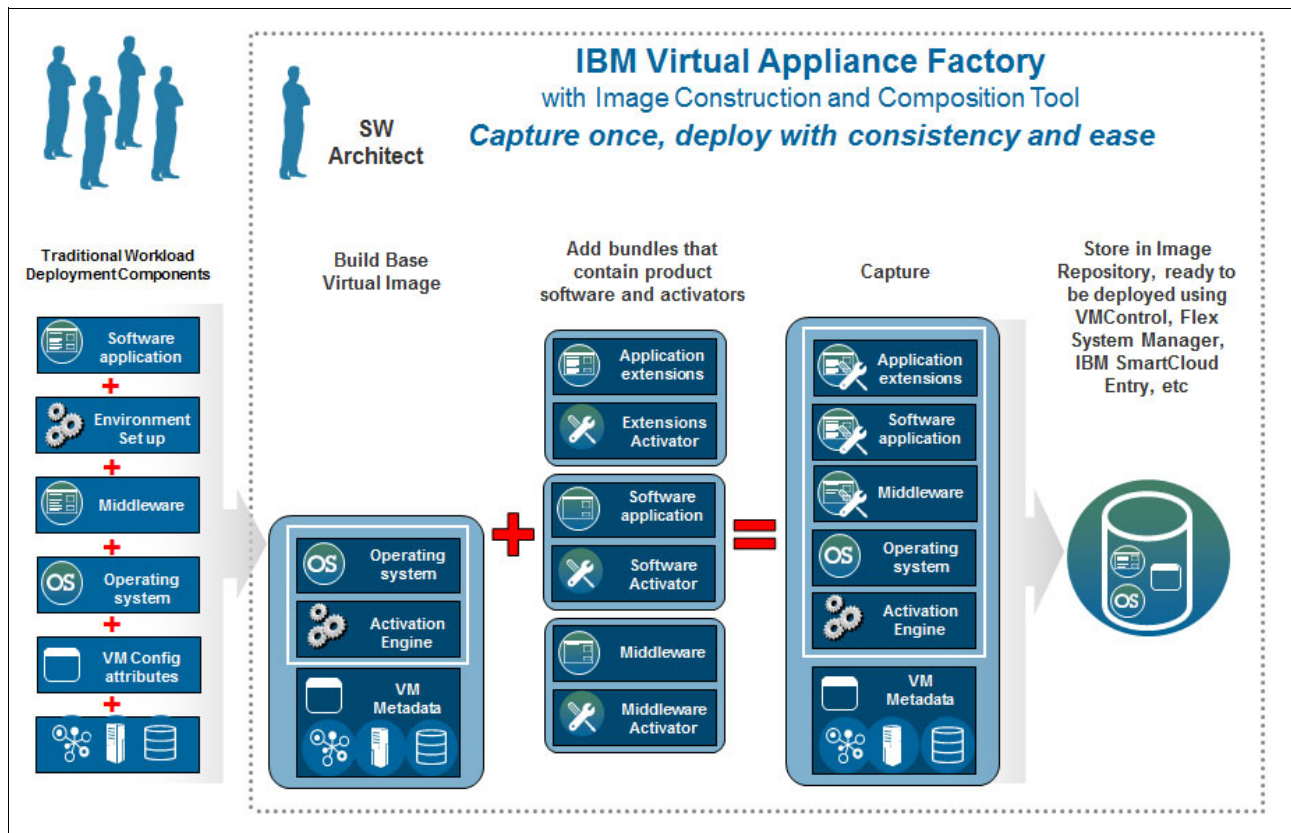


Figure 18-2 IBM i Virtual image process

On the left side of the chart is the traditional deployment of a workload or application. Starting at the bottom, you must allocate the networking, system, and storage resources. Create the virtual machine, install and configure the operating system, install and configure any middleware that is needed, and do any other environment setup that might be required before you approach the business software application. This process can take days or weeks to accomplish.

Virtual appliances are self-contained images with all the software that is needed to run (including the operating system, middleware, and software application), and an activation engine that provides intelligence and configuration activation scripts. MSPs and independent software vendors (ISVs) can provide entire integrated virtual appliances to their customers for quick and easy deployment.

The IBM Virtual Appliance Factory is a proven process. It provides methodology, resources, and tools to create virtual appliances. The process walks through creating a base image, creating software bundles with product-specific activation scripts, and using ICCT to create, synchronize, capture, and export the virtual appliance. This virtual appliance can be used with IBM Systems Director VMControl™, IBM Flex System® Manager™, and IBM SmartCloud® Entry.

18.6 Summary

IBM i is an enterprise-level cloud platform that MSPs and cloud providers can build upon to deploy enterprise private and hybrid cloud infrastructures for their customers. With IBM i 7.1 TR3 and beyond, the technical capabilities are built into the operating system, and key interoperation capabilities with other IBM tools and technologies are available. For more information about this topic, see the IBM i information center at the following website:

<http://pic.dhe.ibm.com/infocenter/iseriess/v7r1m0/index.jsp>

Related publications

The publications that are listed in this section are considered suitable for a more detailed description of the topics that are covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- ▶ *Advanced Functions and Administration on DB2 Universal Database for iSeries*, SG24-4249
- ▶ *Discovering MySQL on IBM i5/OS*, SG24-7398
- ▶ *e-business Globalization Solution Design Guide: Getting Started*, SG24-6851
- ▶ *IBM DB2 Web Query for i Version 2.1 Implementation Guide*, SG24-8063
- ▶ *IBM Technology for Java Virtual Machine in IBM i5/OS*, SG24-7353
- ▶ *Preparing for and Tuning the SQL Query Engine on DB2 for i5/OS*, SG24-6598
- ▶ *RPG: Exception and Error Handling*, REDP-4321
- ▶ *Security Guide for IBM i V6.1*, SG24-7680
- ▶ *Stored Procedures, Triggers, and User-Defined Functions on DB2 Universal Database for iSeries*, SG24-6503
- ▶ *Using IBM DB2 for i as a Storage Engine of MySQL*, SG24-7705
- ▶ *Who Knew You Could Do That with RPG IV? A Sorcerer's Guide to System Access and More*, SG24-5402

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Other publications

These publications are also relevant as further information sources:

- ▶ Tuohy, *The Programmer's Guide to iSeries Navigator*, MC Press, 2005, ISBN 1583470476
- ▶ Tuohy, *Re-engineering RPG Legacy Applications*, Midrange Computing, 1991, ISBN 1583470069
- ▶ Woodbury, *IBM i Security Administration and Compliance*, MC Press, 2012, ISBN 1583473734
- ▶ Yantzi, et al, *The Remote System Explorer*, MC Press, 2008, ISBN 1583470816

Online resources

These websites are also relevant as further information sources:

- ▶ DB2 Connect family
<http://www.ibm.com/software/products/en/db2-connect-family>
- ▶ Globalize your business
<http://www.ibm.com/software/globalization/>
- ▶ IBM and Zend PHP - Overview
<http://www.ibm.com/systems/power/software/i/php/>
- ▶ IBM i 6.1 documentation
http://www.ibm.com/support/knowledgecenter/ssw_ibm_i_61/welcome.html
- ▶ IBM i 7.1 documentation
http://www.ibm.com/support/knowledgecenter/ssw_ibm_i_71/welcome.html
- ▶ IBM i welcome page
http://www.ibm.com/support/knowledgecenter/ssw_ibm_i/welcome
- ▶ IBM Lab Services
http://www.ibm.com/systems/services/labservices/platforms/labservices_power.html
- ▶ IBM Systems Magazine
<http://www.ibmssystemsmag.com/ibmi/>
- ▶ IBM Systems Magazine IBM i Developer blog by Susan Gantner and Jon Paris
<http://ibmsystemsmag.blogs.com/idevelop/>
- ▶ IBM Toolbox for Java and JOpen
<http://www.ibm.com/systems/power/software/i/toolbox/index.html>
- ▶ Integrated Web Services for IBM i
<http://www.ibm.com/systems/power/software/i/iws/>
- ▶ iProDeveloper
<http://www.iprodeveloper.com>
- ▶ ITJungle
<http://www.itjungle.com>
- ▶ PHP: Documentation
<http://www.php.net/docs.php>
- ▶ Rational CafeRD Power
<http://ibm.co/1fSNcET>
- ▶ Rational Developer for i
<http://www.ibm.com/software/products/en/dev-ibm-i/>
- ▶ Rational Developer for Power Systems Software
<http://pic.dhe.ibm.com/infocenter/iadthelp/v8r5/index.jsp>
- ▶ Rational Team Concert
<https://jazz.net/products/rational-team-concert/>

- ▶ RPG Cafe
<http://ibm.co/NorH2U>
- ▶ The Unicode Consortium
<http://www.unicode.org>

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



Redbooks

Modernizing IBM i Applications from the Database up to the User Interface and Everything in Between

(1.0" spine)

0.875" x 1.498"

460 <-> 788 pages



Modernizing IBM i Applications from the Database up to the User Interface and Everything in Between



Learn about preferred practices for modern development

Use modern tools for a modern world

Incorporate data-centric programming for success

This IBM Redbooks publication is focused on melding industry preferred practices with the unique needs of the IBM i community and providing a holistic view of modernization. This book covers key trends for application structure, user interface, data access, and the database.

Modernization is a broad term when applied to applications. It is more than a single event. It is a sequence of actions. But even more, it is a process of rethinking how to approach the creation and maintenance of applications. There are tangible deliveries when it comes to modernization, the most notable being a modern user interface (UI), such as a web browser or being able to access applications from a mobile device. The UI, however, is only the beginning. There are many more aspects to modernization.

Using modern tools and methodologies can significantly improve productivity and reduce long-term cost while positioning applications for the next decade. It is time to put the past away. Tools and methodologies have undergone significant transformation, improving functionality, usability, and productivity. This is true of the plethora of IBM tools and the wealth of tools available from many Independent Solution Providers (ISVs).

This publication is the result of work that was done by IBM, industry experts, and by representatives from many of the ISV Tool Providers. Some of their tools are referenced in the book. In addition to reviewing technologies based on context, there is an explanation of why modernization is important and a description of the business benefits of investing in modernization. This critical information is key for line-of-business executives who want to understand the benefits of a modernization project. This book is appropriate for CIOs, architects, developers, and business leaders.

**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**

SG24-8185-00

ISBN 073843986X