

IBM Express Runtime V2.1

The starting point for learning IBM
Express Runtime V2.1

In-depth information on all
supported platforms

Many useful examples



Aleksandr Nartovich
Sougat Ghosh
Richard Johnson
Kwang Sik Kang
Kimberly D Price
Gaetano Rinciari
Marcelino Villanueva

Redbooks



International Technical Support Organization

IBM Express Runtime V2.1

May 2005

Archived

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

First Edition (May 2005)

This edition applies to Version 2.1 of IBM Express Runtime.

© Copyright International Business Machines Corporation 2005. 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	vii
Trademarks	viii
Preface	ix
The team that wrote this redbook.	ix
Become a published author	xi
Comments welcome.	xi
Part 1. Exploring IBM Express Runtime V2.1	1
Chapter 1. Introduction to Express Runtime V2.1	3
1.1 Application server model.	4
1.2 Middleware products	4
1.3 Why IBM Express Runtime?	5
1.4 Benefits of Express Runtime V2.1	5
1.5 Express Runtime V2.1 architecture.	6
1.5.1 Middleware software	6
1.5.2 Applicable applications	7
1.6 Express Runtime Usage Scenarios	7
1.6.1 Express Runtime for solution development (embed)	7
1.6.2 Express Runtime for middleware only (pre-req)	8
1.6.3 Express Runtime or Express Runtime Plus	8
Chapter 2. What's new in IBM Express Runtime V2.1?	11
2.1 Renamed from Integrated Runtime to Express Runtime.	12
2.2 Build on the latest market leading middleware components	12
2.2.1 IBM HTTP Server 6.0 and IBM WebSphere HTTP plug-in	12
2.2.2 WebSphere Application Server Express V6.0.	12
2.2.3 DB2 Universal Database Express Edition Version 8.2	12
2.3 Added functional support – Enterprise JavaBeans and Java Messaging Services. ...	13
2.4 Console for Express Runtime	13
2.5 Enhanced development and deployment tools	13
2.5.1 Deployment Wizard.	15
2.6 Extended platform support	16
2.7 Express Runtime V2.1 purchase options	16
2.7.1 Passport channel	16
2.7.2 Express Runtime Plus growth path.	17
Chapter 3. Express Runtime architecture	19
3.1 Express Runtime delivered through OEM channel	20
3.1.1 The development environment	20
3.1.2 How does Express Runtime in an OEM scenario work?	22
3.2 Express Runtime delivered through PPA channel	26
3.2.1 How does Express Runtime in a Passport Advantage scenario work?	27
3.3 Summary	29
Part 2. Implementing Express Runtime V2.1	31
Chapter 4. Installing Express Runtime	33
4.1 Planning for the installation	34

4.2 System prerequisites.	34
4.2.1 Hardware requirements.	35
4.3 Installation scenarios.	37
4.3.1 Discussion of the scenarios	37
4.4 Installing the product.	37
4.5 Exploring resulting Express Runtime V2.1 file directory and software location.	57
4.5.1 Windows	57
4.5.2 Linux and Linux on IBM Power	58
4.6 Uninstalling the product.	58
4.6.1 Keep in mind when you uninstall	62
Chapter 5. What's a wrapper?	63
5.1 Exploring a wrapper	64
5.1.1 Application wrapper	64
5.1.2 Solution wrapper	65
5.1.3 Conclusion	65
5.2 Developing a wrapper	65
5.2.1 Developing an application wrapper.	65
5.2.2 Developing a solution wrapper	77
5.3 Support Framework API	86
5.4 Support for other languages	87
Chapter 6. Developing a wrapper	91
6.1 Overview of the sample applications.	92
6.1.1 Trade6.	92
6.1.2 WebFacing application	92
6.2 Developing the Trade6 wrapper for Windows	93
6.2.1 Creating the Trade6 application project	93
6.2.2 Response file (properties file)	110
6.2.3 Creating Trade6 user programs	113
6.2.4 Completing the Trade6 application project	127
6.2.5 Creating Trade6 solution project.	132
6.2.6 Building the Trade6 solution	143
6.3 Developing Wrappers for Trade6 for Linux on POWER	148
6.3.1 Preparing for Trade6 deployment	150
6.3.2 Code customization before deployment	150
6.3.3 Deploying the Trade6 Solution	150
6.3.4 Files required for Trade6 application.	151
6.4 Developing a wrapper for the WebFacing application.	152
6.4.1 Before developing a wrapper	152
6.4.2 Creating an application project	152
6.4.3 Developing user programs	154
6.4.4 Creating the solution project	159
6.4.5 Generating and testing the solution	161
6.5 Debugging user programs.	162
Chapter 7. Packaging a solution	165
7.1 Creating the solution package for Trade6.	166
7.2 Creating installation CDs for a solution	169
7.3 Packaging a solution with just the application.	175
Chapter 8. Deploying a solution	177
8.1 Supported platforms	178
8.1.1 System requirement	178

8.2 Terminology used in this chapter	181
8.3 Deployment methods	181
8.4 Installing the IBM Installation Agent (IIA)	183
8.4.1 Installing on Windows, Linux, and Linux on POWER	184
8.4.2 Installing on i5/OS	190
8.5 Deployment scenarios	195
8.5.1 Scenario 1: Deploying to a local system	196
8.5.2 Scenario 2: Deploying from CDs	207
8.5.3 Scenario 3: Deploying to the remote systems (two systems)	209
8.5.4 More complex scenarios	224
8.6 What if you have one component installed?	224
8.6.1 Distributed platform	224
8.6.2 i5/OS (OS/400) platform	226
8.7 Validating the deployment	226
8.7.1 Log files	226
8.8 Troubleshooting deployment	230
8.8.1 Tracing on the target system	232
Chapter 9. Managing Express Runtime	233
9.1 Console for Express Runtime	234
9.1.1 Terminology	234
9.2 Deploying Console for Express Runtime	234
9.2.1 Example of the remote deployment	234
9.3 Using Console for Express Runtime	239
9.3.1 Logon to Console for Express Runtime	239
9.3.2 Adding a server to the console	241
9.3.3 Starting or stopping a server	243
9.3.4 Viewing log files	243
9.3.5 Backing up databases	244
9.4 Other ways to manage Express Runtime middleware	246
9.4.1 IBM HTTP Server 6.0 on Windows and Linux	246
9.4.2 IBM HTTP Server on OS/400	246
9.4.3 IBM WebSphere Application Server - Express Version 6	247
9.4.4 DB2 UDB Express 8.2 for Windows and Linux	247
9.4.5 DB2 UDB for iSeries	248
9.4.6 Web Administration for iSeries	249
Chapter 10. Migrating wrappers to Express Runtime V2.1	251
10.1 Moving projects and deployment package files	252
10.2 Updating wrapper files	253
10.2.1 Generating the solution	254
Part 3. Appendixes	255
Appendix A. Source code for Trade6 user programs and script files on Windows	257
The application.xml file	258
The solution.sxml file	260
The TradeWinMain.java program	264
The TradeWinPDC.java program	275
The TradeWinCommon.java program	282
The TradeNLSKeys.java program	285
The TradeMessagesNLS.java program	286
The CheckAppInstall.jacl script file	288
The WebSphereConfigProcs.jacl script file	289

The WebSphereScript.jacl script file	312
The Table.ddl file	316
The DB2Script.bat file	318
The SetupProcs.jacl script file	319
The Trade.prop file	321

Appendix B. Source code for Trade6 user programs and script files for Linux on POWER

323

The application.xml file	324
The solution.xml file	326
The TradeLnxMain.java program	326
The TradeLnxPDC.java program	344
The TradeLnxCommon.java program	353
The CheckAppInstall.jacl script file	357
The WebsphereConfigProcs.jacl script file	358
The WebSphereScript.jacl script file	381
The DB2Script.sh script file	386
The Table.ddl response file	388
The SetupProcs.jacl script file	389
The Trade.prop script file	391

Appendix C. Source code for Flight400 user programs and script files on OS/400 .

The application.xml file	396
The solution.xml file	399
The WebSphereScript.jacl file	402
The SamplePDC.java file	404
The SampleMain.java file	412
The SampleExit.java file	423
The SampleCommon.java file	424
The SampleCommands.java file	428

Appendix D. Additional material

Locating the Web material	435
Using the Web material	435
System requirements for downloading the Web material	436
How to use the Web material	436
Additional files to run the sample applications	437

Related publications

IBM Redbooks	439
Online resources	439
How to get IBM Redbooks	439
Help from IBM	440

Index

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

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

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

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

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

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

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates 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. 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 IBM's application programming interfaces.

Trademarks

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

AIX®
AS/400®
DB2®
DB2 Universal Database™
Domino®
@server®
i5/OS™

IBM®
ibm.com®
iSeries™
Lotus®
OS/400®
PartnerWorld®
Passport Advantage®

POWER™
POWER5™
Rational®
Redbooks™
Redbooks (logo) ™
WebSphere®

The following terms are trademarks of other companies:

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. 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.

Intel, Intel Inside (logos), and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

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

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

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

Preface

This IBM® Redbook gives a broad understanding of IBM Express Runtime V2.1. This product is designed to simplify development and deployment of the middleware products along with the J2EE applications.

This product seamlessly integrates:

- ▶ WebSphere® Application Server V6.0
- ▶ IBM DB2® Universal Database V8.2
- ▶ IBM HTTP Server V6.0
- ▶ IBM Web server plug-in
- ▶ Express Runtime Console

This integration considerably saves time and effort for installing these middleware products and J2EE applications on a variety of platforms.

This book is an excellent source of information about IBM Express Runtime V2.1 for I/T developers, architects, and consultants.

The team that wrote this redbook

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

Aleksandr V. Nartovich is a Senior I/T Specialist in the IBM ITSO, Rochester Center. He joined the ITSO in January 2001 after working as a developer in the IBM WebSphere Business Components organization. During the first part of his career, Aleksandr was a developer in AS/400® communications. Later, he shifted his focus to business components development on WebSphere. Aleksandr holds two degrees: one in Computer Science from the University of Missouri-Kansas City and the other in Electrical Engineering from Minsk Radio Engineering Institute. You can reach Aleksandr by sending e-mail to: alekn@us.ibm.com

Sougat Ghosh is an IBM Consultant and has over 7 years experience in information systems planning, project management, business analysis, and technical pre-sales. Sougat joined IBM India in 2001 and has been responsible for several product installations including the first DB2 EE integration in AP. Sougat did his Post-Graduation from IIM-Calcutta and holds many certifications. After having worked for 2 years as a product specialist across various products and technologies, he is now responsible for enablement for various technology areas, including skill and technology enablement within India.

Richard Johnson is a Senior IT Specialist for the IBM Production Introduction Centre, based at IBM Hursley Park, UK. In this customer facing role he promotes the successful introduction of new product releases through initiatives such as Early Programs, skills transfer workshops and field enablement. Richard is skilled in a wide variety of products and technologies including SOA, Web Services, J2EE, IBM Workplace, Eclipse, and WebSphere products in general. He also has four years experience as a consultant for IBM Software Services for WebSphere, designing and implementing solutions for IBM clients and business partners across EMEA and worldwide. He holds a Masters degree in Chemistry from the University of Oxford.

Kwang Sik Kang serves as a Senior IT Architect with IBM's Software Services for WebSphere. He leads the architecture and development efforts for Fortune 500 clients. He has been published in over 30 industry periodicals, has several patents pending, and has spoken at conferences across the nation, including FTP's Java™ Pro Live! Kwang is pursuing an MS in Computer Science from NTU at Walden University and holds a BS in Computer Science from SUNY Binghamton.

Kimberly D Price is a Technical Support Software Engineer at the IBM Innovation Center for Business Partners in Dallas, Texas within IBM Software Group in the United States. She supports and enables ISVs on the Express Runtime product through QA, partner assistance, Beta support, webcast delivery, and product distribution. Previously, she worked for the WebSphere Portal/Lotus® WorkPlace technical support team and was the Web developer for the intranet site, w3.developer.ibm.com.

Gaetano Rinciari is an Advisory I/T Specialist since 1999 with IBM Global Services in IBM Italy. He works for Integrated Technologies Services in the Delivery Services area. He is a certified WebSphere Application Server administrator and an AIX® / Linux® specialist. He provides support to local customers in the banking and insurance market segments.

Marcelino Villanueva is an Advisory I/T Specialist with IBM Global Services in the Philippines. He provides technical support for a diverse set of clients in banking, insurance, government, education, manufacturing, and telecommunication industries. He is an ASEAN Tower Specialist supporting IBM @server iSeries™ customers across the region. He conducts high-level Account Management practices for the top customers in his assignment. He has been a technical resource and executed project management tasks on iSeries Smoothstart services. He teaches IBM courses in the Philippines on all areas of iSeries administration. Before joining IBM Global Services in 1997, Marcelino has worked in the AS/400 Partners in Development Center assisting ISVs and BPs in porting front-end applications to access the native database of the AS/400 platform. His areas of expertise include Java, OS/400®, Windows® NT, Client Access, Domino® on iSeries, and TCP/IP. He is a Sun Certified Java Programmer (SCJP), Microsoft® Certified Systems Engineer (MSCE), Microsoft Certified Professional plus Internet (MCP+i), IBM Certified Specialist - IBM @server iSeries System Administrator, and an IBM Certified Specialist - IBM @server iSeries Technical Solutions Implementer for V5R2 and V5R1.

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

Shih-in Bick
IBM Schaumburg, Illinois

Brad Fawcett
Doug Fiesel
Kevin Hubbard
Kyle Henderson
Bonnie Lodermeier
Mark Osteraas
Michael Payne
Oko Swai
Ronald F Persik
Sahdev P Zala
IBM Rochester, Minnesota

Arshad Bahl
IBM Somers, New York

Barnaby Court
Kelley Greeson
Johanna Cook
Jordan Liggitt
Erich Magee
Hiren Patel
Bob Sizemore
IBM Raleigh, North Carolina

Peggy Danner
IBM Dallas, Texas

Yvonne Lyon
IBM International Technical Support Organization, San Jose, California

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- Send your comments in an email to:

redbook@us.ibm.com

- Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. JLU Building 107-2
3605 Highway 52N
Rochester, Minnesota 55901-7829

Archived



Part 1

Exploring IBM Express Runtime V2.1

In this part of the book, we provide an overview of IBM Express Runtime V2.1, its architecture, and the new features in Version 2.1. This is the “executive overview” of the product, but is very important for understanding the development and deployment topics described in the second part of the book.

Archived



Introduction to Express Runtime V2.1

This chapter provides a high level overview of IBM Express Runtime V2.1.

1.1 Application server model

To match business and customer needs, Web initiatives such as J2EE applications have become a required part of customer shops. To support these applications, customers adopt the new architecture called an application server model. This architecture is based on a multi-tier approach. The basic model is shown in Figure 1-1.

There are 4 typical tiers in this model:

- ▶ Client tier
- ▶ Web server tier
- ▶ Application server tier
- ▶ Enterprise Information Systems (EIS) tier

There are separate products that are deployed at each tier. It can be a large investment in buying these products and training special personnel to maintain this architecture.

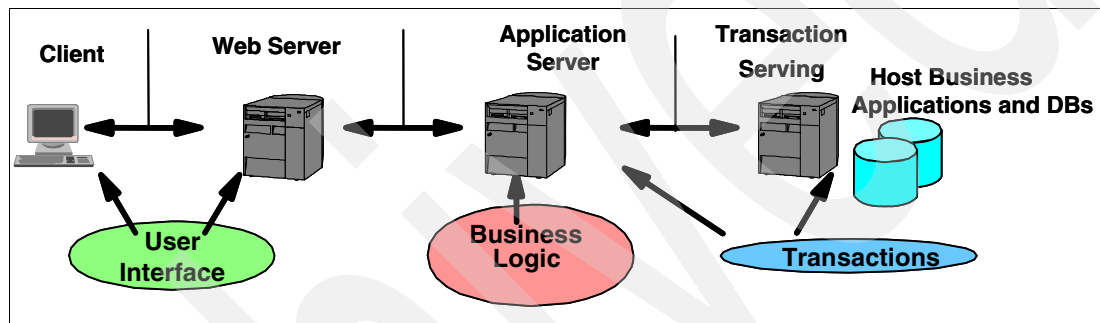


Figure 1-1 Application server model

1.2 Middleware products

To expand on Figure 1-1, we will now discuss software products matching each tier of the application server architecture:

- ▶ At the client tier, a Web browser is the most widely used interface.
- ▶ A Web server supports the communication. IBM provides the IBM HTTP Server that maps into this tier.

However, a Web server doesn't provide a framework for running business applications. It is done at the application server tier. A Web server plug-in component plays an important role in enabling the Web server to pass clients' requests to the application server and support a mechanism for managing the workload in a distributed environment.

IBM supplies the Web server plug-in component that supports the most popular Web servers on the market.

- ▶ An application server provides the environment where we can run a business logic and/or access the backend systems and databases in a uniform fashion.

WebSphere Application Server or WebSphere Portal Server is a good example of the IBM products supporting this tier.

- ▶ Finally, at the EIS tier, we can run a wide range of legacy systems and databases. IBM DB2 Universal Database™ is a premier product with a rich set of functions that the customers implement at the EIS tier.

Figure 1-2 shows the mapping of the IBM products into the application server architecture.

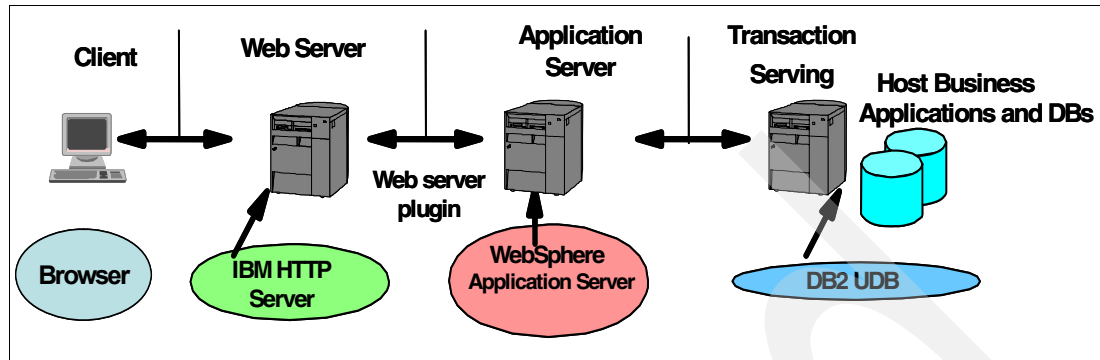


Figure 1-2 Web Architecture with IBM products

1.3 Why IBM Express Runtime?

Most companies implemented this Web architecture within their IT infrastructure. Small to medium businesses (SMBs) or departments within larger organizations have resource limitations when implementing IT infrastructure. They have to evaluate constantly changing technology within limited restrictions. Although an issue at any company, hardware, software, planning, development, and maintenance are more taxing at an SMB. SMB companies need flexible, integrated, and portable customer solutions to streamline planning and development phases. The maintenance phase is more efficient when dealing with one supplier, contract, license and support structure. All three phases reduce time, effort, and resource bandwidth, thereby reducing inefficiencies and optimizing profitability.

IBM implemented this idea in the IBM Express Runtime product. It bundles the most critical middleware products supporting the application server model in a single installation and configuration process that dramatically simplifies the deployment and maintenance of the J2EE applications.

1.4 Benefits of Express Runtime V2.1

IBM has recognized the SMB needs in the marketplace to utilize a synchronized effort through an affordable middleware infrastructure from which they can enable their customer's business applications. Additionally, the framework provides a complete business solutions that are easily deployable to customer environments.

Express Runtime V2.1 has provided additional functionality to the previous version. Here are some benefits of the Express Runtime V2.1 infrastructure and framework:

- ▶ Built with market leading Express products designed especially for the mid-market
- ▶ Rapid return on investment:
 - Competitive pricing versus purchasing the individual IBM software products
 - Flexible pricing and licensing
 - Reduced time to market with highly integrated and configured Express product, out of the box, with solution, application, and middleware templates
 - Integrated administration and comprehensive enablement rewards providers and customers who take advantage of the broader Express portfolio

- ▶ Development and deployment:
 - One single installation that includes IBM HTTP Server, WAS Express, and DB2 UDB Express, along with any ISV/RSI applications
 - Function-rich development tools based on the Eclipse platform
 - Significant technical support via quick start samples and documentation
 - Pre-integrated and pre-configured solutions that can be deployed directly onto customer systems
 - Open standards based solutions and infrastructure
 - Multiple platform support
- ▶ Administration:
 - Single administration console to manage all middleware
 - Integrated administration via browser interface remote access for administration
 - A consistent experience makes it easier for solution providers to access the information, tools, financing, and support they need to work with IBM's Express Runtime V2.1
 - Delivering enterprise level security to reduce risk without adding complexity to mid-market customers and Solution Providers
- ▶ Ease of doing business:
 - Simplified approach to leveraging the IBM middleware portfolio
 - One product, one license, single contract, resulting in an easy upgrade and support structure
- ▶ Growth:
 - Customers who grow can trade up to Express Runtime Plus without a reinstallation (see 1.6.3, "Express Runtime or Express Runtime Plus" on page 8)

1.5 Express Runtime V2.1 architecture

Express Runtime is a critical element of the IBM offerings that provides a complete solution to integrate configuration and installation. Express Runtime streamlines the process to create solutions. Each solution is adaptable to meet specific customer requirements. Whether only one or all of the middleware components are necessary, Express Runtime provides a single IBM product license and a single point of contact for service. Each deployable solution is comprised of two component types:

- ▶ Middleware software
- ▶ Applicable applications

These components are tied together in a single installation package. The deployment wizard makes the process of installing and configuring these components as simple as possible.

IBM Express Runtime V2.1 supports a number of operating systems. See 4.2, "System prerequisites" on page 34 for more information about the supported platforms.

1.5.1 Middleware software

The middleware components included in Express Runtime V2.1 are:

- ▶ IBM HTTP Server 6.0
- ▶ IBM WebSphere Application Server (WAS)- Express 6.0

- ▶ IBM DB2 Universal Database Express Edition for Windows and Linux 8.2
- ▶ IBM Web server plug-in

1.5.2 Applicable applications

As well as middleware software, solutions may also include one or more applications.

If you develop a Web application, Express Runtime V2.1 includes a development environment to streamline creation of the J2EE applications (based on Java servlets and JavaServer Pages) and integrating them with the middleware components.

After development completes, you will package middleware components and any applicable applications into a solution.

Express Runtime V2.1 uses the following tools for solution development and deployment:

- ▶ **Express Runtime Developer:** An Eclipse-based plug-in for solution development, including installation and configuration through custom editors. Express Runtime helps you offer a complete solution, including integrated installation and configuration.
- ▶ **IBM Rational® Web Developer:** An Eclipse-based full featured Integrated Development Environment (IDE). It is designed for building, testing, and deploying Java and Java 2 Platform, Enterprise Edition (J2EE) applications. It provides integrated development support for building J2EE applications with HTML pages, servlets, and JavaServer Pages (JSP).
- ▶ **Deployment Wizard:** The deployment wizard installs a solution on one or more target systems. The deployment wizard offers an interface that helps you deploy a solution task-by-task.
- ▶ **Console for Express Runtime:** This is a standalone Web-based console which provides the administrative interface to all middleware components that are parts of IBM Express Runtime V2.1. This support is available across multiple machines and/or platforms.

1.6 Express Runtime Usage Scenarios

IBM Express Runtime V2.1 is available through two programs:

- ▶ Original Equipment Manufacturer (OEM)
- ▶ Passport Advantage® (PPA)

1.6.1 Express Runtime for solution development (embed)

Express Runtime is available through the OEM program for Solution Providers who develop and sell business solutions that contain integrated IBM middleware software provided with Express Runtime. The OEM programs provide attractive discount or price incentives as revenue volume increases with or without volume commitments.

An OEM agreement is implemented via a direct contract between IBM and the Solution Provider. Under this contract, IBM provides the Solution Provider a master copy of the Express Runtime software, which is copied and integrated within the Solution Provider's software application into their solution(s). Solution sales are reported back to IBM and tracked for attained incentives. IBM provides telephone defect support to the Solution Provider for the Express Runtime licenses covered under Maintenance.

OEM enables Solution Providers to offer their end customers an integrated solution powered by IBM middleware via a sole contact — the Solution Provider. The end customer can be comfortable knowing that their solution is powered by IBM proven middleware, yet continue to have the attentive care of their solution provider.

1.6.2 Express Runtime for middleware only (pre-req)

Express Runtime is also available through the standard Passport Advantage IBM programs. These are simple, comprehensive programs that cover software license acquisition and maintenance options under a single, common set of agreements, processes, and tools.

IBM Express Runtime through Passport Advantage programs target Solution Providers that resell or pre-require Express Runtime to their solution(s). In the Passport Advantage programs, the End Customer licenses the Express Runtime directly from IBM. Furthermore, while the Express Runtime entitlement's maintenance is up to date, IBM provides direct support to the End Customer.

1.6.3 Express Runtime or Express Runtime Plus

IBM Express Runtime has a limit of four processors for deploying WebSphere Application Server - Express and DB2 UDB Express (see Table 1-1). IBM Express Runtime Plus is the growth path to Express Runtime without the need to perform any migration steps. Express Runtime Plus offers license terms which increases the limit of processors permitted to be used (see Table 1-1).

Table 1-1 Express Runtime and Express Runtime Plus

	Express Runtime		Express Runtime Plus	
Scenario 1: Two Separate Servers	WAS on server 1 DB2 on server 2	1-2 Processors 1-2 Processors	WAS on server 1 DB2 on server 1	3-4 Processors 3-4 Processors
Scenario 2: One Server for both	WAS and DB2 on same server	1-4 Processors	WAS and DB2 on same server	5-8 Processors

Each of the Express Runtime offerings is available under two separate charge units:

- ▶ Per processor
- ▶ Per user

IBM Express Runtime V2.1

Per processor licence:

- ▶ This is available in units of one.
- ▶ It is limited to a maximum of two processors per server.
- ▶ For configurations with a single server, the number of processor entitlements that you need to purchase is equal to the number of processors that will be used by the installed program.
- ▶ For configurations with multiple servers, the number of processor entitlements that you need to purchase is equal to the highest number of processors used by either DB2 UDB Express V8.2 or WebSphere Application Server — Express V6.0.
- ▶ Internet users (connections outside the end-user's enterprise) are permitted.

Per user licence:

- ▶ This is available in units of one, except that the initial order must be for a minimum of five entitlements; subsequent orders may be in units of one.
- ▶ It is limited to a maximum of two processors per server.
- ▶ You must purchase an entitlement for each user of either (or both) DB2 UDB Express V8.2 or WebSphere Application Server - Express V6.0.
- ▶ Internet users (connections outside the end-user's enterprise) are not permitted.

IBM Express Runtime Plus V2.1


Per processor licence:

- ▶ This is available in units of one.
- ▶ It is limited to a maximum of four processors per server.
- ▶ For configurations with a single server, the number of processor entitlements that you need to purchase is equal to the number of processors that will be used by the installed program.
- ▶ For configurations with multiple servers, the number of processors entitlements that you need to purchase is equal to the highest number of processors used by either DB2 UDB Express V8.2 or WebSphere Application Server - Express V6.0.
- ▶ Internet users (connections outside the end-user's enterprise) are permitted.

Per user licence:

- ▶ This is available in units of one, except that the initial order must be for a minimum of five entitlements; subsequent orders may be in units of one.
- ▶ It is limited to a maximum of four processors per server.
- ▶ You must purchase an entitlement for each user of either (or both) DB2 UDB Express V8.2 or WebSphere Application Server - Express V6.0.
- ▶ Internet users (connections outside the end-user's enterprise) are not permitted.

Archived



What's new in IBM Express Runtime V2.1?

In this chapter, we introduce changes and feature enhancements in Express Runtime V2.1.

2.1 Renamed from Integrated Runtime to Express Runtime

In this version, there was a product name change to Express Runtime. Previously, in version 1.1, it was called Integrated Runtime (IR).

2.2 Build on the latest market leading middleware components

Express Runtime V2.1 has been built on leading middleware components. New middleware components are:

- ▶ IBM HTTP Server 6.0
- ▶ IBM WebSphere HTTP plug-in
- ▶ WebSphere Application Server Express V6.0
- ▶ DB2 Universal Database Express Edition Version 8.2

2.2.1 IBM HTTP Server 6.0 and IBM WebSphere HTTP plug-in

As included in Express Runtime V2.1, the full IBM HTTP Server services multiple Web requests and interacts with the application server. The IBM HTTP Server provides a security-rich, standards-based Web environment capable of handling high volume transactions quickly.

More product information on IBM HTTP Server 6.0 is available at the following Web sites:

- ▶ For distributed platforms, refer to:
<http://www-306.ibm.com/software/webservers/httpservers/>
- ▶ For OS/400, refer to:
<http://www.ibm.com/servers/eserver/series/software/http>

The WebSphere HTTP plug-in provides help in automating the configuration of the IBM HTTP Server and the WebSphere Application Server-Express. This automated configuration reduces the time and resources required to establish and configure a secure, fully functional Web environment.

2.2.2 WebSphere Application Server Express V6.0

Express Runtime contains the execution environment for WebSphere Application Server Express V6.0. This provides a standards-based application server that supports J2EE 1.4 and extensions, including Servlets, JSPs, EJB, JMS, and Web Services.

For more product information on WebSphere Application Server Express V6.0, see:

<http://www-306.ibm.com/software/webservers/appserv/express/>

2.2.3 DB2 Universal Database Express Edition Version 8.2

Execution components from DB2 UDB Express provide a fully functional, standards-based relational database specifically designed to support midsize business applications.

For more product information on DB2 Universal Database Express Edition Version 8.2, see:

<http://www-306.ibm.com/software/data/db2/udb/edition-express.html>

2.3 Added functional support – Enterprise JavaBeans and Java Messaging Services

With WebSphere Application Server Express V6.0, Express Runtime V2.1 supports Enterprise JavaBeans (EJBs) and Java Messaging Services (JMS) messaging.

2.4 Console for Express Runtime

The new console for Express Runtime provides a remote, Web-based administration for managing all of the middleware components within the solution, from a single point. Figure 2-1 shows the Welcome tab page for the Integrated Solutions Console for Express Runtime. For instance, some practical applications here would be adding users, changing passwords, and evaluating log files. The Console for Express Runtime is described in detail in 9.1, “Console for Express Runtime” on page 234.

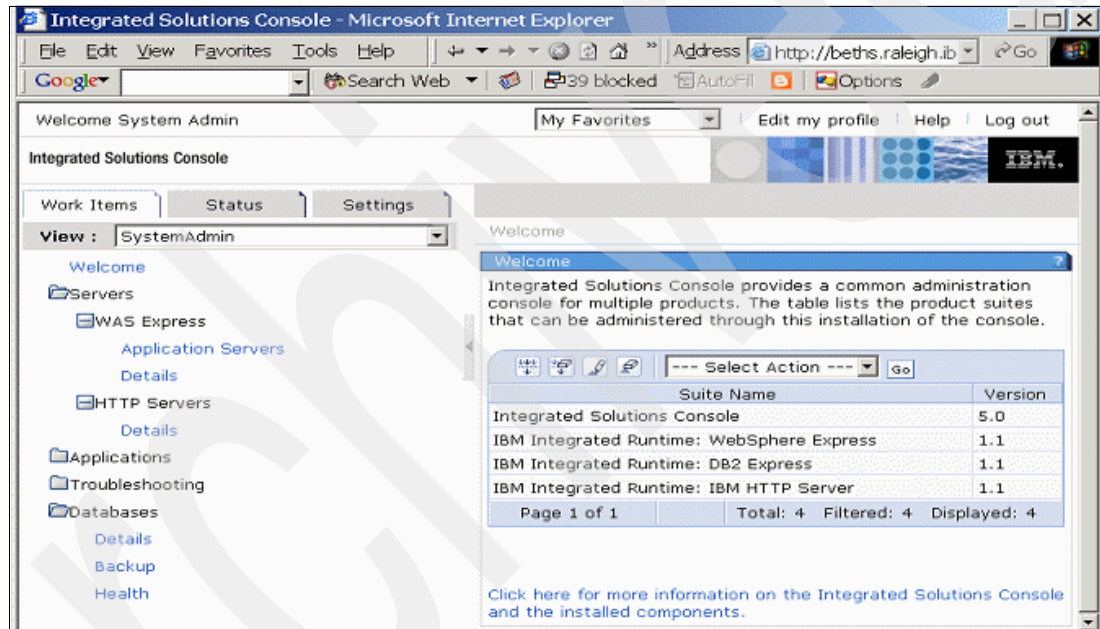


Figure 2-1 Welcome page for Integrated Solutions Console

2.5 Enhanced development and deployment tools

Express Runtime V2.1 includes the redesigned development and deployment tools. The development tool, Express Runtime Developer, is based on Rational Web Developer V6.0. As a result, it is recommended that you develop your Web application and integrate it into a solution using Express Runtime Developer.

Express Runtime Developer includes multiple wizards for creating and modifying the applications and solutions:

- ▶ Creating the application and solution projects
- ▶ Importing and exporting the solutions
- ▶ Creating a CD(s) or DVD(s) for solution deployment
- ▶ Modifying the application and solution XML files
- ▶ And many more...

In addition, the special purpose editors for working with the XML files have been added to Express Runtime Developer. Now, instead of working with the XML source, developers use the Application or Solution Wrapper Editor. Figure 2-2 and Figure 2-3 demonstrate the new Application Wrapper Editor.

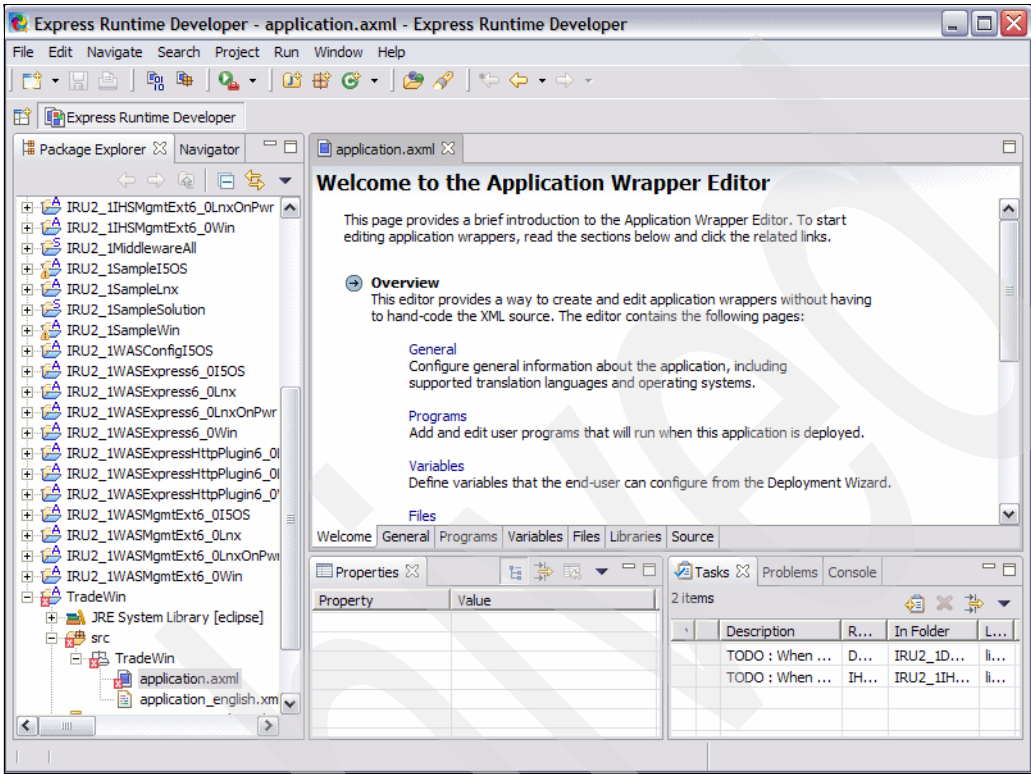


Figure 2-2 Application Wrapper Editor Welcome tab page

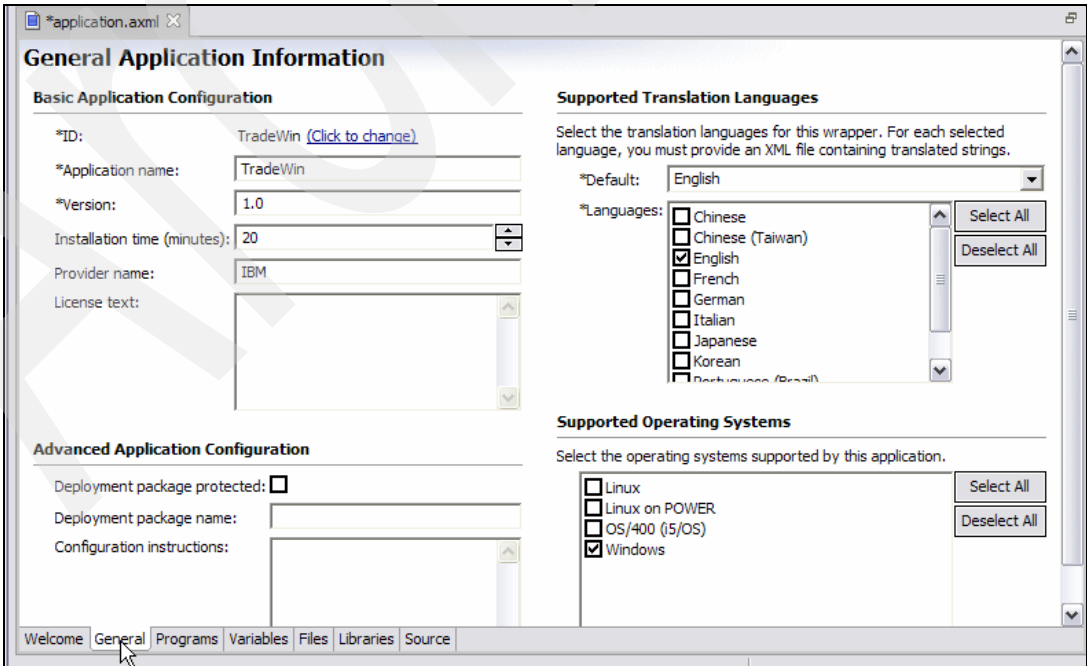


Figure 2-3 General application information

2.5.1 Deployment Wizard

Previously called Solution Deployer, the Deployment Wizard tool allows for users to deploy a solution locally or remotely. Figure 2-4 and Figure 2-5 demonstrate the new Deployment Wizard look and feel.

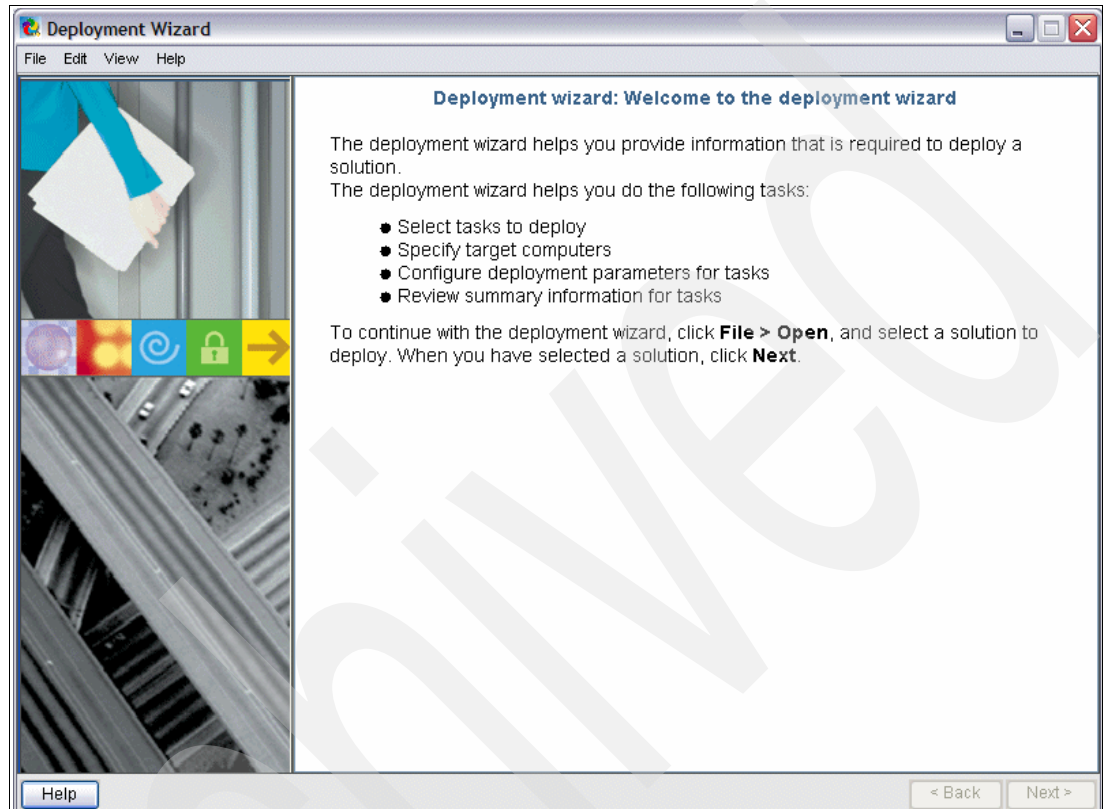


Figure 2-4 Welcome window to the Deployment Wizard

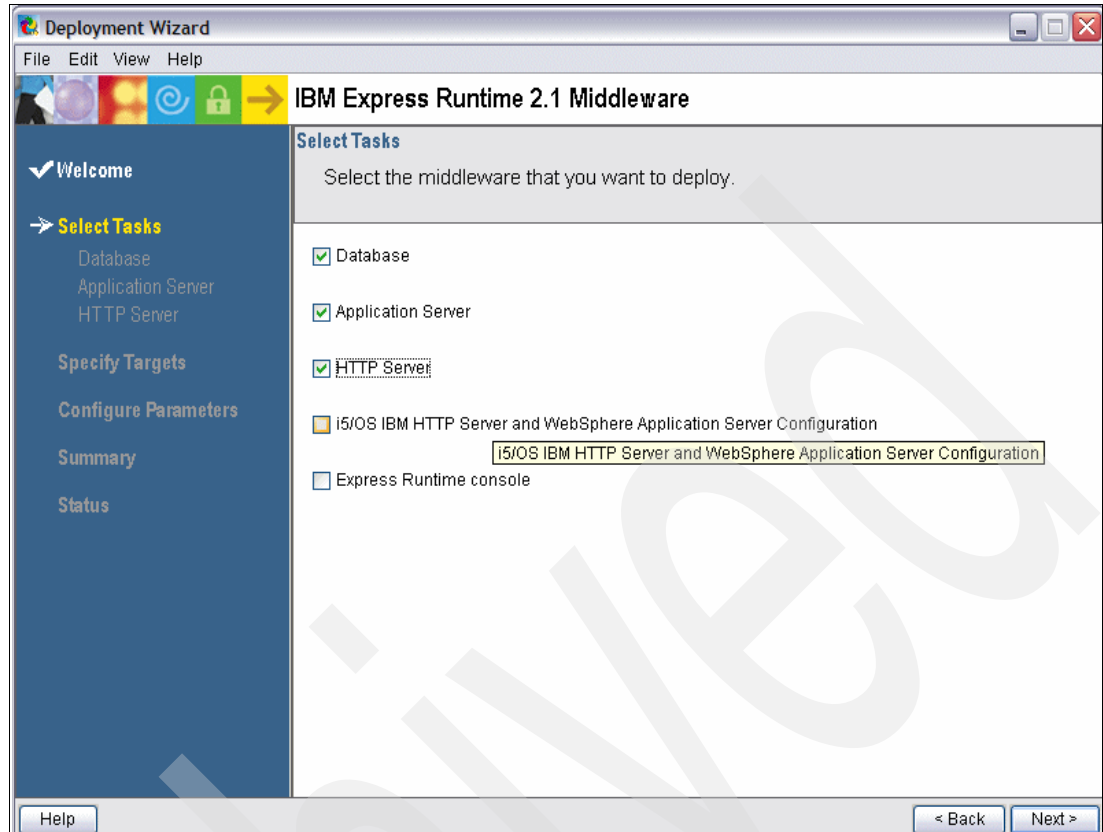


Figure 2-5 Select Tasks window of the Deployment Wizard

2.6 Extended platform support

Platform support has been expanded to include:

- ▶ Linux On Power, using SUSE LINUX Enterprise Server 9.0 or Red Hat Enterprise Linux AS 3.0 (deployment target only)
- ▶ Red Hat Enterprise Linux (RHEL) 3.0 WS/AS/ES
- ▶ Suse Linux Enterprise 8.0(deployment target only), 9.0
- ▶ Red Flag Advanced Server 4.1 (deployment target only)
- ▶ IBM i5/OS or OS/400 V5R2 or V5R3 (deployment target only)

2.7 Express Runtime V2.1 purchase options

There are several purchase options with Express Runtime V2.1. In this section we describe these options.

2.7.1 Passport channel

As noted in 1.6, “Express Runtime Usage Scenarios” on page 7, Express Runtime continues to be available through the OEM programs. However, beginning with V2.1, the Express Runtime and Express Runtime Plus are also available through the Passport Advantage programs.

2.7.2 Express Runtime Plus growth path

The Express Runtime Plus offering was developed to provide you with a growth path when your business requires more processing power. The Express Runtime Plus license permits you to run on more processors per server to increase your processing power (see Table 1-1 on page 8).

Archived

Express Runtime architecture

IBM Express Runtime V2.1 helps address many of the problems faced by Independent Software Vendors (ISVs) and Small/Medium-sized Businesses (SMBs) related to obtaining, installing, configuring, and managing middleware components.

In this chapter, the reader will become familiar with the two scenarios for the Express Runtime:

- ▶ Express Runtime delivered through the Original Equipment Manufacturer (OEM) channel programs
- ▶ Express Runtime delivered through the Passport Advantage (PPA) channel programs

Each section details the architecture, component composition, development, and deployment process for its respective scenarios.

3.1 Express Runtime delivered through OEM channel

In the first scenario, a fictitious retailer, XYZ Inc., has engaged an ISV to provide it with a total solution consisting of hardware, software, and a Web-centric application. Prior to Express Runtime, the ISV had to allocate time not only to develop the application, but also a significant amount of time to obtain middleware components from multiple vendors, then install and configure those components. In addition, the ISV would have to perform this task at each customer site that bought their solution.

With Express Runtime, the ISV can use provided tools to develop the application and package it with the necessary middleware components in a single, complete solution.

The solution can be stored on a set of Compact Discs (CDs), Digital Video Discs (DVDs), or on a staging server, and then be deployed to the customer's environment. The ISV not only avoids the frustration of obtaining middleware components from multiple vendors, but also saves time in installing, configuring, and supporting those components.

3.1.1 The development environment

The Express Runtime provides a powerful set of development tools. In a development environment, the following components are installed (see Figure 3-1).

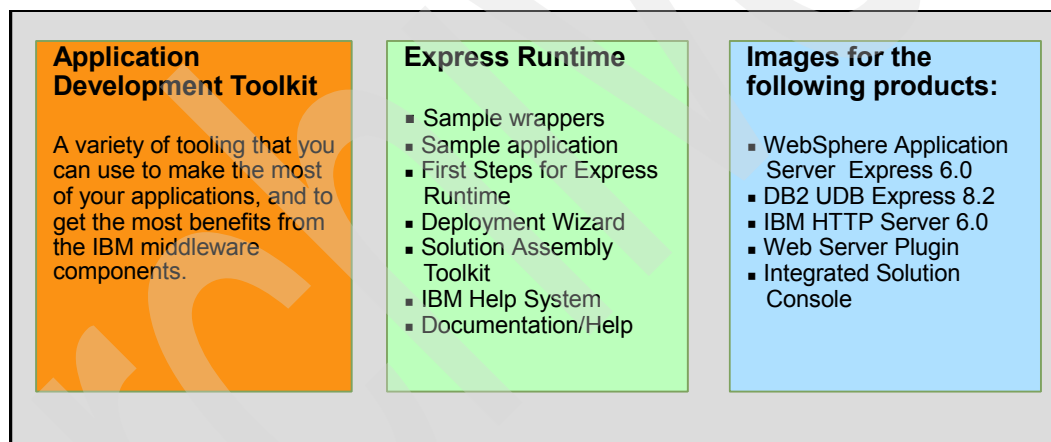


Figure 3-1 Express Runtime components installed on a development platform

Application development tools

Express Runtime V2.1 includes a set of development tools to simplify the process of developing a J2EE application and integrating it with the middleware components.

Rational Web Developer

This is an integrated development environment (IDE) for building Web-centric applications. The Rational Web Developer may be used for developing applications to be deployed on WebSphere Application Server - Express and for deployment packages.

Application Development Toolkit

The Application Development Toolkit enables you to deploy the application development tools, like Rational Web Developer, to your PC.

Express Runtime

The install preloads the Express Runtime Developer workspace with all the sample wrappers and documentation necessary to build a complete Express Runtime solution. An IBM Help facility is available along with additional Express Runtime help documentation.

Compressed product images

The compressed images of the IBM middleware components are stored on the development environment and are installed only during the deployment process.

IBM WebSphere Application Server - Express V6.0

This provides a standards-based J2EE application server that supports servlets/Java ServerPages (JSPs), Enterprise JavaBeans (EJBs), Java Messaging Service (JMS), and easily ties into enterprise information systems (EISs), such as relational databases.

IBM DB2 Universal Database Express Edition V8.2

This provides a fully functional, standards-based relational database designed to support medium business applications.

IBM HTTP Server 6.0

This provides a fast, secure standards-based Web environment.

IBM Web Server plug-in

This helps automate the configuration of the IBM HTTP Server and WebSphere Application Server - Express, which greatly reduces the time to establish and configure a secure, functional Web environment.

IBM Integrated Solutions Console

This provides consolidated and remote administration of the solution's middleware components (that is, IBM WebSphere Application Server Express, IBM DB2 UDB Express, and IBM HTTP Server) through a central Web-based console.

3.1.2 How does Express Runtime in an OEM scenario work?

Figure 3-2 shows the six-step process for developing and deploying a solution in Express Runtime environment.

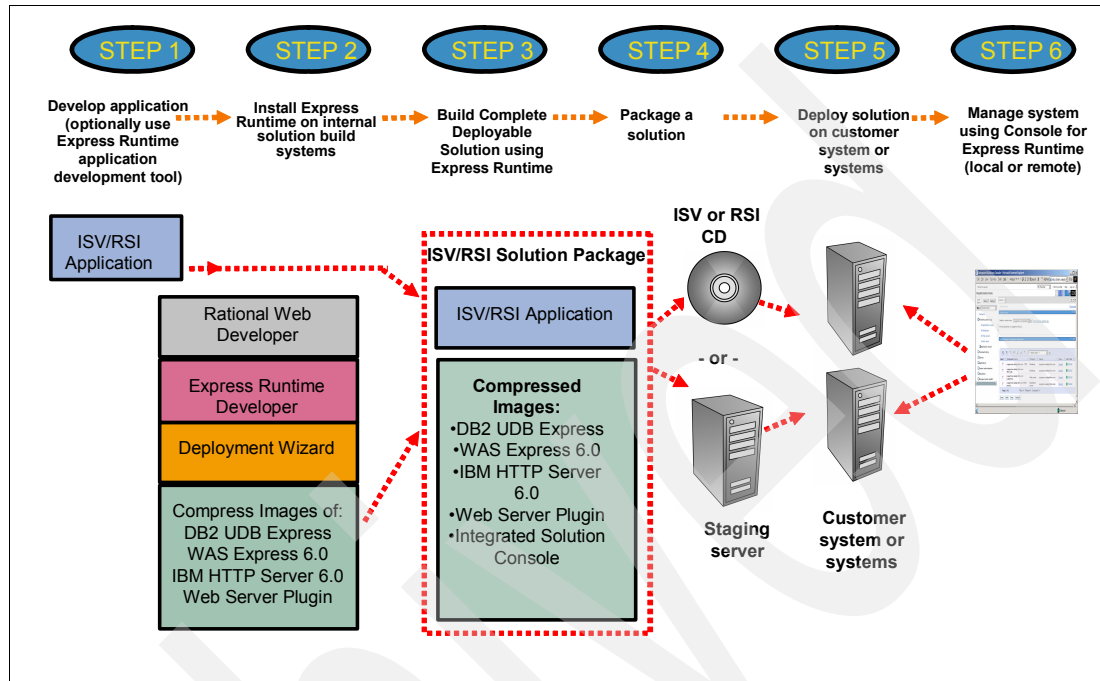


Figure 3-2 Express Runtime development and deployment process

Step 1: Develop ISV/RSI application

The ISV can use Rational Web Developer that is included in Express Runtime, to build, test, and deploy Web-centric, Web Services, and J2EE-centric applications (Rational Web Developer supports a subset of J2EE specification).

Alternately, the ISV can choose to build an application using its own application development tools. However, the ISV still has to use Express Runtime Developer to package it with the middleware components.

Step 2: Install Express Runtime

The ISV installs the Express Runtime onto the development environment (see Figure 3-3).

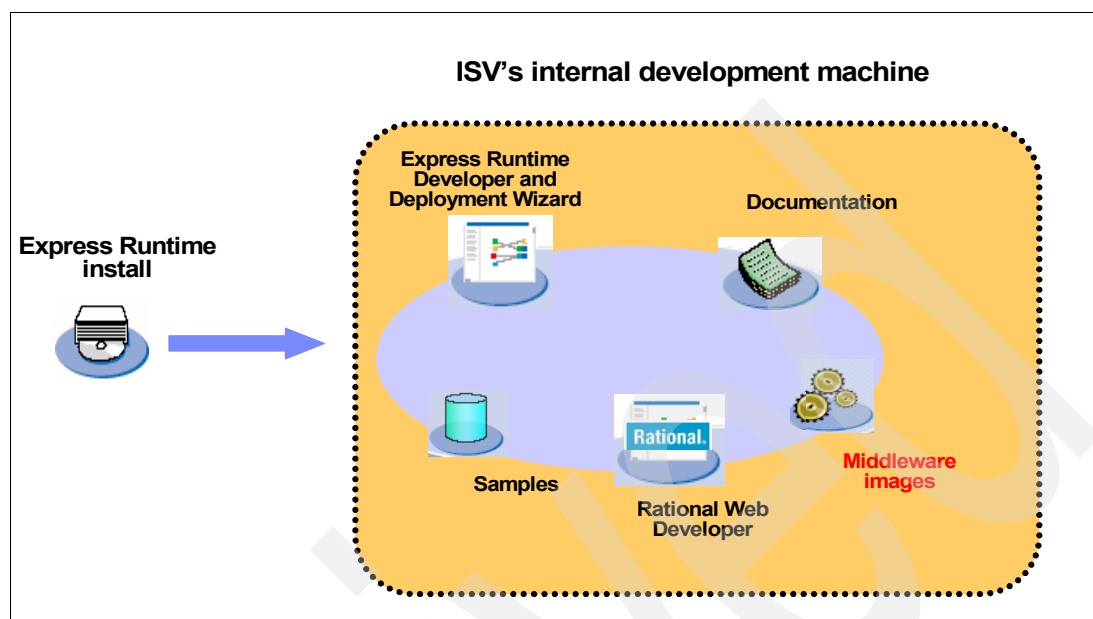


Figure 3-3 Express Runtime install onto the ISV's internal development environment

Step 3: Build a complete solution with Express Runtime Developer

The ISV integrates the application with the required middleware components. The result of this development effort is an ISV solution, which:

- ▶ Includes all the required components necessary to run the ISV's application (see Figure 3-4)
- ▶ Uses a single install process for all components, including the ISV's application.

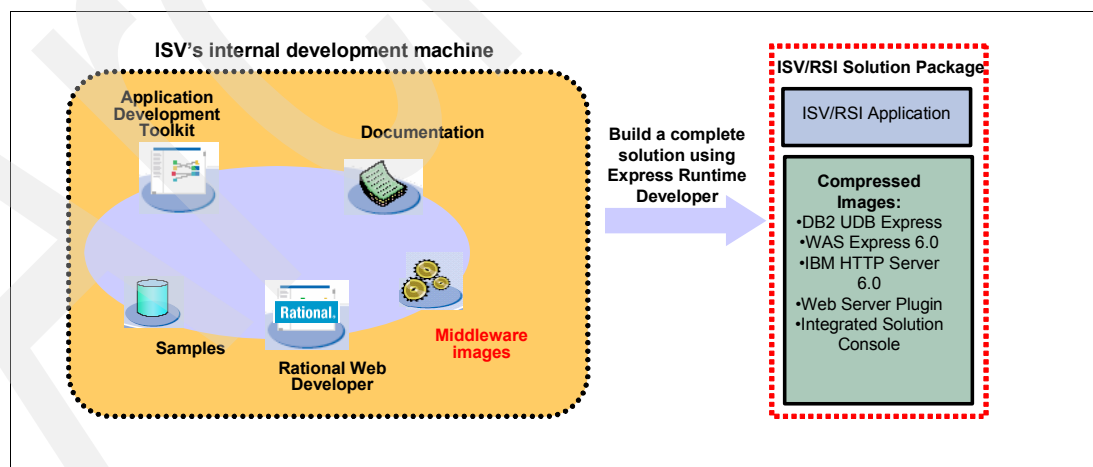


Figure 3-4 Build a complete solution using the Express Runtime Developer

Express Runtime uses wrappers to integrate an ISV's application with other components, like IBM's middleware. For each application, an application wrapper is used to check for prerequisites, drive the install process, and perform post-install configuration activities, such as creating JDBC resources in WebSphere Application Server Express, establishing port configurations, or creating database tables.

A solution wrapper ties together all the applications (application wrappers) that make up the solution. Creating the application and solution wrappers is the main development effort in building a complete solution. The wrappers do not replace the install technology for the middleware components. Instead the wrapper is used to pass deployment time information (such as hostname) to the middleware install technology.

Step 4: Package a solution

Once wrapper development is complete, the wrappers along with the compressed images of the middleware components and your application(s), if so desired, may be stored on CDs, DVDs, or on a staging server for later deployment to the customer's environment (see Figure 3-5).

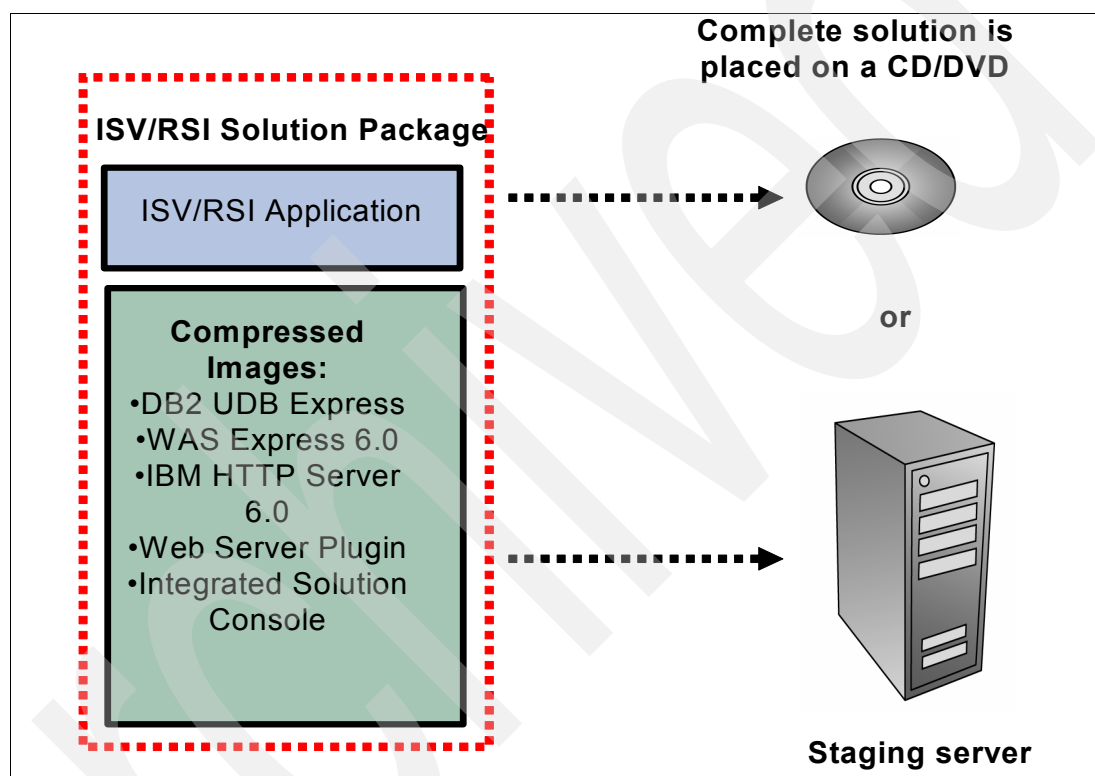


Figure 3-5 Package the complete ISV solution onto a set of CDs, DVDs or a staging server

Step 5: Deploy solution on customer environment

There are two distinct options for deploying a solution:

- ▶ From CD(s)/DVD(s)
- ▶ From a Staging Server

In both cases, the Deployment Wizard will drive the deployment process.

In the first approach, the solution is packaged on a set of CDs/DVDs. The deployer will load the CDs/DVDs into the target machine and the Solution Launcher will start the installation from the CD/DVD. It gathers the information required to complete the deployment and then starts the Deployment Wizard.

In the second approach, the solution is packaged on a staging server to drive the deployment.

Local and remote deployment are supported with both options.

In remote deployment, the solution can be deployed to multiple systems. For example, the implementation of a solution might dictate deployment of the HTTP Server to one system and DB2 UDB Express, WebSphere Application Server Express, and the ISV application to another system (see Figure 3-6).

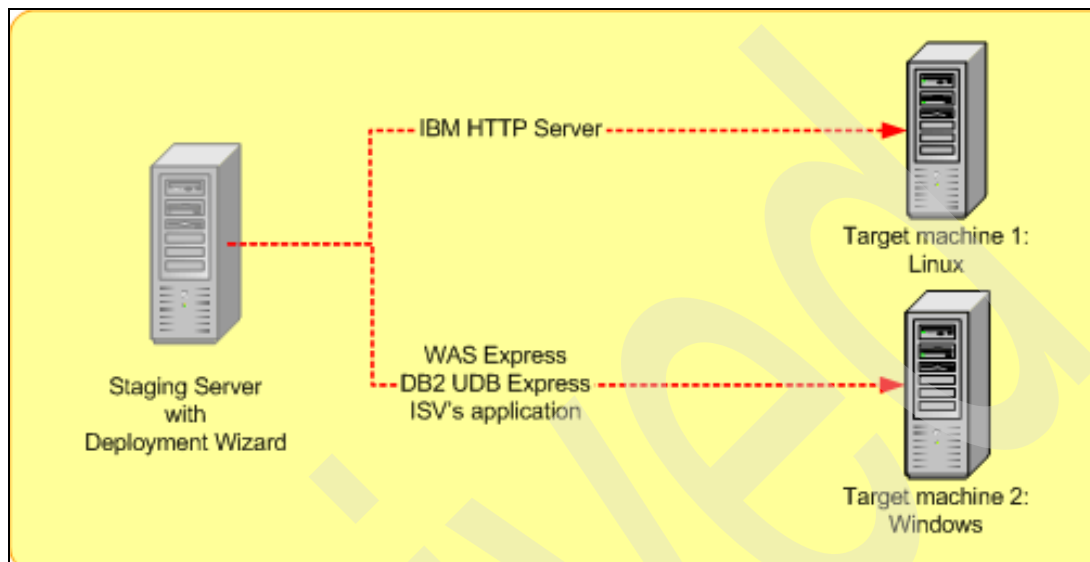


Figure 3-6 Deploy ISV solution to multiple systems with option to partition middleware components

To perform a remote deployment, the end customer system must be installed with the *IBM Installation Agent (IIA)*, provided with Express Runtime. It is used to:

- ▶ Establish communication between the staging server and the target system.
- ▶ Drive the installation process.

Step 6: Manage system using provided console

The solution and its middleware components can be administered from *console for Express Runtime*. Administrators can manage multiple instances of the WebSphere Application Server Express, DB2 UDB Express, and IBM HTTP Server.

The console is based on IBM's *Integrated Solution Console (ISC)*. Express Runtime plugs a set of elements that allow the administration of the Express Runtime components. See the ISC sample page in Figure 3-7.

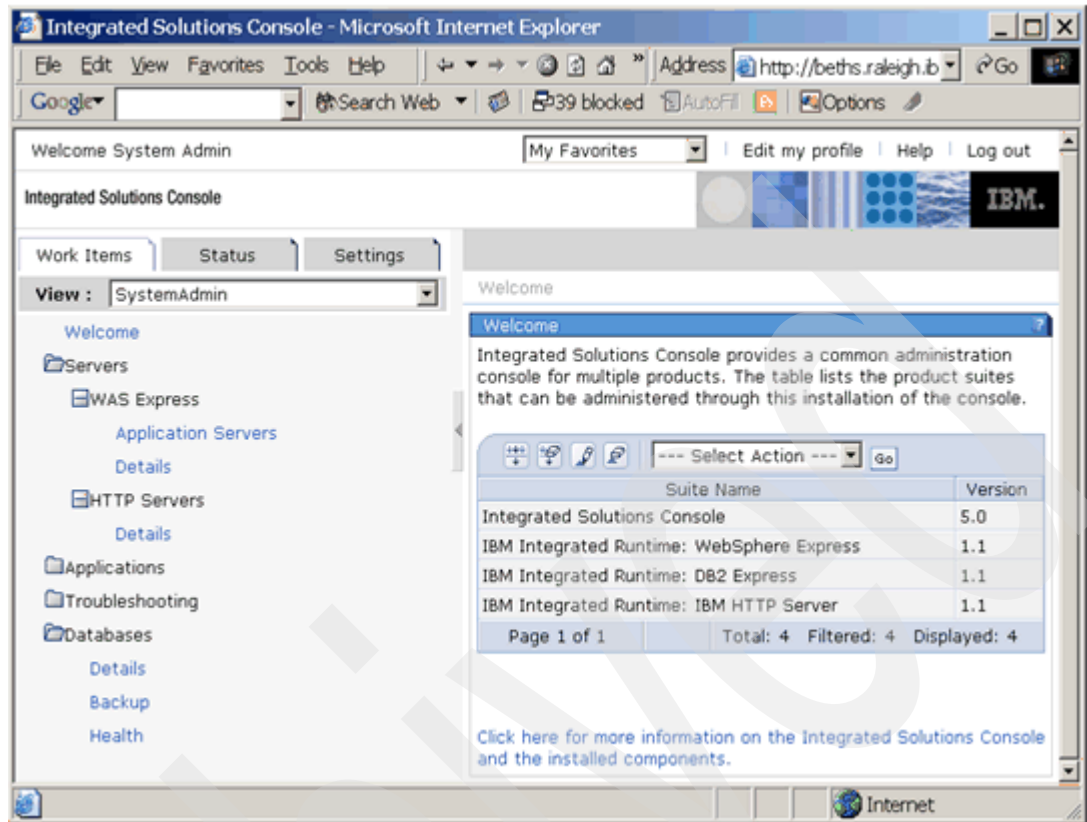


Figure 3-7 Manage the system using the Integrated Solutions Console's Web-based interface

3.2 Express Runtime delivered through PPA channel

In the second scenario, a fictitious retailer, XYZ Inc., has engaged a Regional System Integrator (RSI) to provide a total solution consisting of hardware, software, and a Web-centric application. The RSI typically does not develop applications; instead, it enables a company to use off-the-shelf hardware and software packages. A Business Partner (BP) will focus on developing the application.

Instead of spending time obtaining middleware components from multiple vendors, the RSI pre-requires Express Runtime to complete its solution and yet still have a single installation of all the necessary middleware components.

The Express Runtime provides a single interface that presents the user with the choice of middleware components to deploy. The middleware components are deployed onto the customer environment directly from the Express Runtime media. The rest of the solution applications and technologies can be installed outside the Express Runtime to work with the middleware.

3.2.1 How does Express Runtime in a Passport Advantage scenario work?

Since the application and the middleware components are not packaged into one complete solution like in the Express Runtime OEM channel scenario, the RSI will perform an alternate series of steps for development and deployment as shown in Figure 3-8.

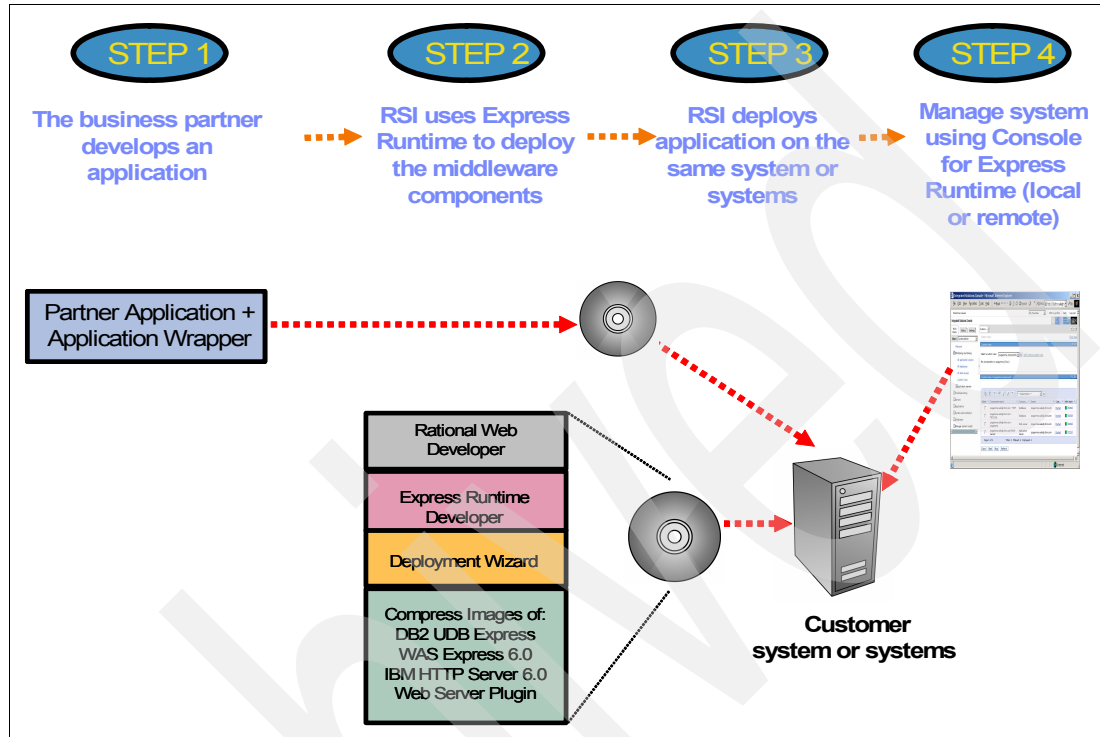


Figure 3-8 Express Runtime's development and deployment process

Step 1: BP develops an application

Business Partners may use their own application development tools to build their applications. Alternatively, the Business Partner may use on behalf of their customer the customer's copy of Express Runtime to develop any application (within the licensed restricted use) for production use by the customer.

They develop a wrapper for this application to make it deployable in Express Runtime environment.

Step 2: RSI deploys middleware components

The RSI can choose to either store the compressed images of the middleware components on a customer's staging server or deploy them from media directly onto the customer environment (see Figure 3-9).

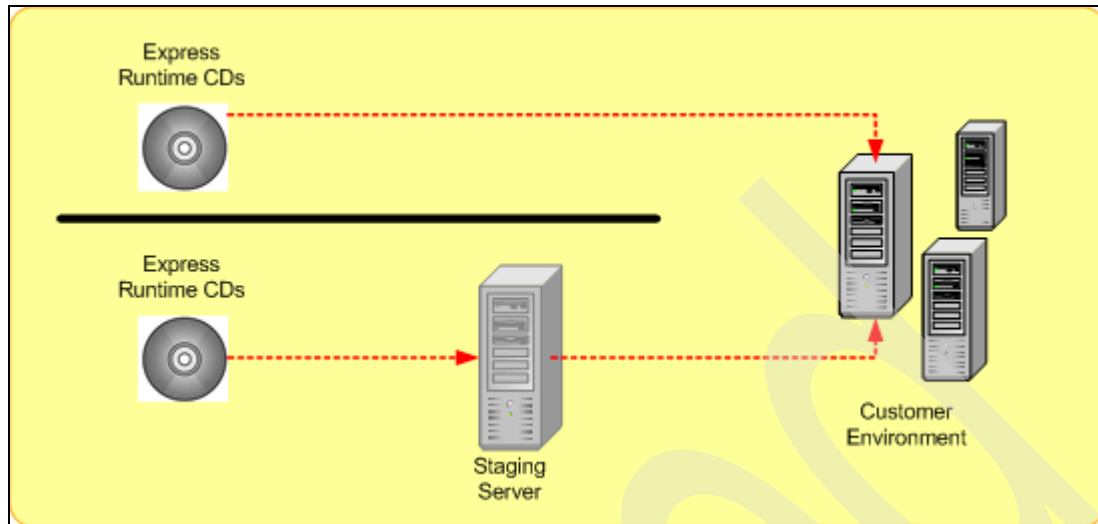


Figure 3-9 Deploy middleware components from Express Runtime CDs or staging server onto customer's environment

Step 3: RSI deploys application

After the middleware components are deployed to the customer environment, the RSI can then deploy the partner's application (see Figure 3-10).

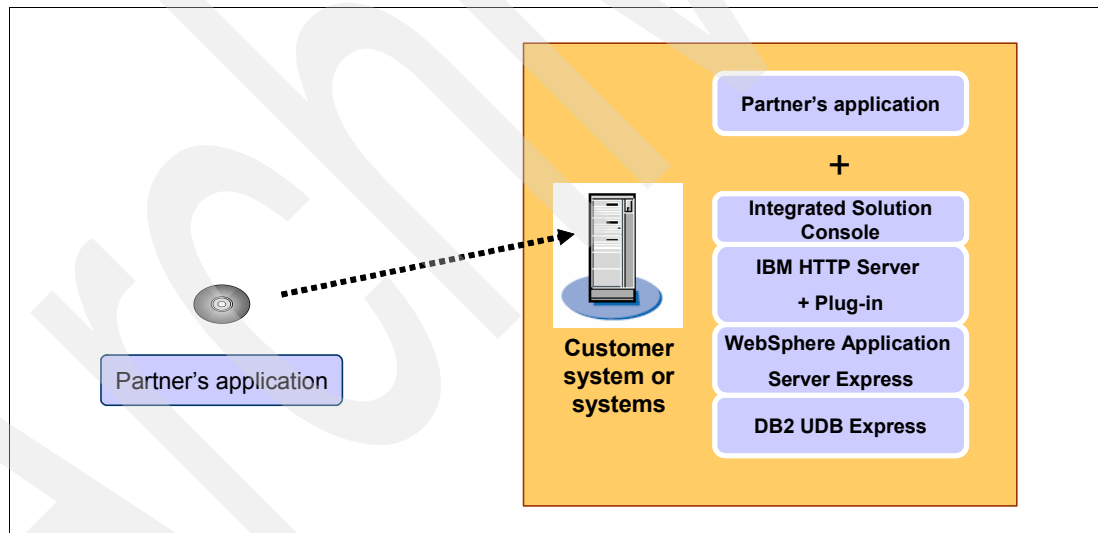


Figure 3-10 Deploy the application to the customer's prepared environment

Step 4: Manage system using provided console

The solution and its middleware components can be administered from *console for Express Runtime* (see Figure 3-7 on page 26). Administrators can manage multiple instances of the WebSphere Application Server Express, DB2 UDB Express, and IBM HTTP Server.

3.3 Summary

The IBM Express Runtime provides the fundamental middleware infrastructure necessary to deliver your mid-market applications. It contains a fully functional execution environment powered by IBM's market-leading middleware products.

- ▶ IBM WebSphere Application Server Express 6.0 provides a robust application server platform.
- ▶ IBM DB2 Universal Database Express 8.2 provides a relational database management system.
- ▶ IBM HTTP Server provides a secure Web environment.

The Express Runtime delivery channel programs govern the usage of the Express Runtime components and how the components are packaged and delivered. There are certain limitations on usage of these middleware components:

- ▶ Under an OEM agreement, an ISV integrates its application with Express Runtime middleware components in a single, but complete solution package. As such, the middleware components are restricted to be used *only* to support the bundled application.
- ▶ Under a PPA license, a customer may use the Express Runtime components to support multiple applications.

Archived



Part 2

Implementing Express Runtime V2.1

In this part of the book, we provide detailed instructions for:

- ▶ Installing Express Runtime V2.1
- ▶ Developing a wrapper
- ▶ Packaging a solution
- ▶ Managing Express Runtime V2.1

Archived

Installing Express Runtime

This chapter contains installation information for the Express Runtime V2.1 installation option. Depending on the user needs, the following components can be installed through the install wizard:

- ▶ Express Runtime V2.1:
 - Sample wrappers and applications
 - Deployment Wizard
 - Solution Assembly Toolkit
 - Express Runtime First Steps
 - Product Documentation
 - IBM Help System
- ▶ IBM Rational:
 - IBM Rational Web Developer V6.0
 - Rational Product Updater
 - Rational Software Developer Platform

4.1 Planning for the installation

Depending on the usage scenario, your planning process and system requirements can be different. However, a typical planning process includes:

1. Acquiring Express Runtime V2.1 evaluation CDs: Business Partners can access them from either of the following Web sites:
<http://www.ibm.com/partnerworld/expressruntime/member/request.html>
<http://www.developer.ibm.com/welcome/evalsoft.html>
2. Planning to set aside enough time for installation. Depending on your server, the process may take more or less time. With that said, the default typical Express Runtime V2.1 installation takes about an hour.
3. This chapter helps you evaluate the available installation options and determine which options you require.

4.2 System prerequisites

This section outlines the Operating System (OS) requirements for Express Runtime V2.1. Table 4-1 breaks down this information by two criteria:

- By OS
- By the usage scenario

The *Development or Deployment* column specifies:

- If a platform can be used to develop a solution and possibly an application
- If a platform can be used for running Deployment Wizard

The *Deployable as Target OS* column indicates if this platform can be a target system on which you deploy a solution.

Table 4-1 Express Runtime V2.1 Platform Table

Platform	Operating System	Development or Deployment	Deployable as Target OS?
Windows	Windows XP Professional SP 2	both	No
	Windows 2000 Server SP4	both	Yes
	Windows 2000 Advanced Server SP4	both	Yes
	Windows 2000 Professional SP3	both	Yes
	Windows Server 2003, Standard Edition SP1	both	Yes
	Windows Server 2003, Enterprise Edition SP1	both	Yes
Linux (Intel® platforms only)	Red Flag Advanced Server 4.1	deployment only	Yes
	Red Hat Enterprise Linux 3.0 WS/AS/ES	both	Yes
	SUSE LINUX Enterprise Server 8.0	deployment only	Yes
	SUSE LINUX Enterprise Server 9.0	both	Yes

Platform	Operating System	Development or Deployment	Deployable as Target OS?
Linux (IBM POWER5™ processor-based technology systems only)	SUSE LINUX Enterprise Server 8.0	deployment only	Yes
	SUSE LINUX Enterprise Server 9.0	deployment only	Yes
	Red Hat Enterprise Linux AS 3.0	deployment only	Yes
OS/400	V5R2	neither	Yes
i5/OS ¹	V5R3	neither	Yes

¹ The OS/400 operating system is known as the i5/OS™ operating system beginning with V5R3.

4.2.1 Hardware requirements

In this section you can find the hardware requirements for the systems that you plan to use as the *development* platform.

Hardware requirements for the deployable platforms are described in 8.1, “Supported platforms” on page 178.

Windows development requirements

Here are the Windows requirements for an Express Runtime V2.1 installation:

- ▶ A minimum of 512 MB of memory; 1 GB is recommended.
- ▶ At minimum, an Intel Pentium® III class processor with a minimum clock speed of 600 MHz. A Pentium IV class processor with a minimum clock speed of 1.2 GHz is recommended.
- ▶ The following network support must be configured when deploying solutions to network-attached target computers:
 - TCP/IP
 - DNS
- ▶ A local area network (LAN) connection.
- ▶ An SVGA monitor with a minimum 1024 x 768 video resolution configured to display a minimum color depth of 256 colors.
- ▶ Internet Explorer 6.0 SP 1+ Web browser to view the online documentation and readme.
- ▶ Approximately 13 GB of disk space to install and run Express Runtime. Refer to the readme for complete disk space requirements.
- ▶ Administrative authority (required for product installation and uninstallation).
- ▶ Eclipse versions supported based on the following criteria:
 - IES - Eclipse Full SDK - 3.0
 - eclipse.org - Eclipse SDK - 3.0

Linux development requirements

Here are the Linux requirements for an ERV2.1 installation:

- ▶ A minimum of 512 MB of memory; 1 GB is recommended.
- ▶ At minimum, an Intel Pentium III class processor with a minimum clock speed of 600 MHz. A Pentium IV class processor with a minimum clock speed of 1.2 GHz is recommended.

- ▶ The following network support must be configured when deploying solutions to network-attached target computers:
 - TCP/IP
 - DNS
- ▶ A LAN connection.
- ▶ An SVGA monitor with a minimum 1024 x 768 video resolution configured to display a minimum color depth of 256 colors.
- ▶ Any Linux supported video card that supports the resolution requirements.
- ▶ Mozilla 1.4 Web browser to view online documentation and readme.
- ▶ Approximately 13 GB of disk space to install and run Express Runtime. Refer to the readme for complete disk space requirements.
- ▶ Administrative authority (required for product installation and uninstallation).
- ▶ Eclipse versions supported — based on the following criteria:
 - IES - Eclipse Full SDK - 3.0
 - eclipse.org - Eclipse SDK - 3.0

Linux on IBM POWER development requirements

Here are the Linux on IBM POWER™ requirements for an ERV2.1 installation:

- ▶ 2 GB of memory.
- ▶ A RS64-IV processor with a minimum clock speed of 600 MHz.
- ▶ The following network support must be configured when deploying solutions to network-attached target computers:
 - TCP/IP
 - DNS
- ▶ A LAN connection.
- ▶ An SVGA monitor with a minimum 1024 x 768 video resolution configured to display a minimum color depth of 256 colors.
- ▶ Any Linux supported video card that supports the resolution requirements.
- ▶ Mozilla 1.4 Web browser to view online documentation and readme.
- ▶ Approximately 85 MB of disk space to install and run Express Runtime. Refer to the readme for complete disk space requirements.
- ▶ Administrative authority (required for product installation and uninstallation).
- ▶ Eclipse versions supported — based on the following criteria:
 - IES - Eclipse Full SDK - 3.0
 - eclipse.org - Eclipse SDK - 3.0

4.3 Installation scenarios

Express Runtime V2.1 allows for the following installation options:

- ▶ *Install IBM Express Runtime*. This option provides two suboptions:
 - Typical Installation
 - Deployment Wizard Only Installation
- ▶ *Install only IBM Express Runtime middleware components*. This option deploys the chosen middleware components on a selected platform. See “**Install only IBM Express Runtime middleware components**” on page 51, for more information.

4.3.1 Discussion of the scenarios

The four options ultimately result in three different installations:

- ▶ *Typical Installation* is the default decision tree. It installs:
 - Express Runtime Developer with sample applications, solutions, and documentation
 - Optionally, Rational Web Developer
 - Deployment Wizard, which can be used to subsequently install the Express Runtime middleware I (IHS, WAS, DB2)

A typical install path option also allows you to choose the middleware components you want to install for deployment by OS.

Choose this option if you want to wrapper an application to build your own solution.

- ▶ *Deployment Wizard Only Installation* (creating a Staging Server) installs Deployment Wizard and the sample solutions, one of which can subsequently be used to install the Express Runtime middleware. It differs from the typical installation because it does not install the Express Runtime Developer for creating wrappers and solutions. Choose this option if you only want to deploy a solution (either the sample one or one you had previously built).
- ▶ The *Install only IBM Express Runtime middleware components* option leverages the Deployment Wizard to allow you to install the middleware components shipped with Express Runtime locally or to remote machines. Use this option if you only want to install the IBM middleware components (WAS, DB2, IHS).

4.4 Installing the product

Although most of the Express Runtime V2.1 wizard guided install is the same across platforms, the differences are apparent when opening the wizard and getting access to Express Runtime V2.1 after a successful install. Next we will show how to open the wizard in Windows or Linux platforms.

1. First put CD 1 (or DVD 1) into the CD-Drive.

Tip: You can install IBM Express Runtime silently. Use the response file named IRU_setup.iss to set the installation options. You can use the same options that you would select during the interactive installation. The options are set in the response file. Start the silent installation using a command similar to:

```
WindowsSetup -options x:\yyy\...\IRU_setup.iss
```

The WindowsSetup executable program and IRU_setup.iss response file are on disk1 of the IBM Express Runtime CDs. Use the LinuxSetup or LinuxPPCSetup executable programs for Linux platforms. The -options parameter must be the fully-qualified name of the IRU_setup.iss response file. The installation results are logged to the IRU_install.log file in the specified installation directory.

- a. On Windows platforms:

If autorun is enabled, the Launchpad should start automatically. If it doesn't start, double-click **WindowsLaunchpad.exe** in the CD root directory.

- b. On Linux platforms:

In Linux, we have included instructions to run the media from the CD. The installation wizard needs a graphic console or an X11 screen.

In the command window, run the following two commands to mount and open the CD directory:

```
mount /mnt/cdrom
cd /mnt/cdrom
```

Run the following command to display the content of the CD:

```
ls -l
```

The results of this command are shown in Example 4-1.

Example 4-1 Results of the ls command

```
[root@relinux disk1]# ls -l
total 298612
-rwxrwxrwx  1 root    root          38 Jan  9 06:51 autorun.inf
drwxrwxrwx  3 root    root       4096 Jan 10 08:14 IRU_LinuxLaunchpadSetup
drwxrwxrwx  3 root    root       4096 Jan 10 08:14 IRU_LinuxPPCLaunchpadSetup
drwxrwxrwx  3 root    root       4096 Jan 10 08:14 IRU_setup
-rwxrwxrwx  1 root    root        6994 Jan  9 03:38 IRU_setup.iss
-rwxrwxrwx  1 root    root     7085452 Jan  9 03:37 IRU_setup.jar
drwxrwxrwx  3 root    root       4096 Jan 10 08:14 IRU_WinLaunchpadSetup
drwxrwxrwx  4 root    root       4096 Jan 10 08:14 license
-rwxrwxrwx  1 root    root    47742976 Jan  9 04:46 LinuxLaunchpad
-rwxrwxrwx  1 root    root    50897920 Jan  9 07:39 LinuxPPCLaunchpad
-rwxrwxrwx  1 root    root    53522432 Jan  9 07:50 LinuxPPCSetup
-rwxrwxrwx  1 root    root    50380800 Jan  9 04:17 LinuxSetup
-rwxrwxrwx  1 root    root         13 Jan  9 06:30 media.inf
drwxrwxrwx 11 root    root       4096 Jan 10 08:14 readme
drwxrwxrwx  3 root    root       4096 Jan 10 08:14 tools
-rwxrwxrwx  1 root    root    46933232 Jan  9 05:16 WindowsLaunchpad.exe
-rwxrwxrwx  1 root    root    48823684 Jan  9 07:40 WindowsSetup.exe
[root@relinux disk1]#
[root@relinux disk1]# ./LinuxLaunchpad
```

Run the **LinuxLaunchpad** command to start installation.

Note: For Linux on IBM PowerPC, run **LinuxPPCLaunchpad**.

- Figure 4-1 shows the initial LaunchPad panel. Select **Install** to begin with installation.

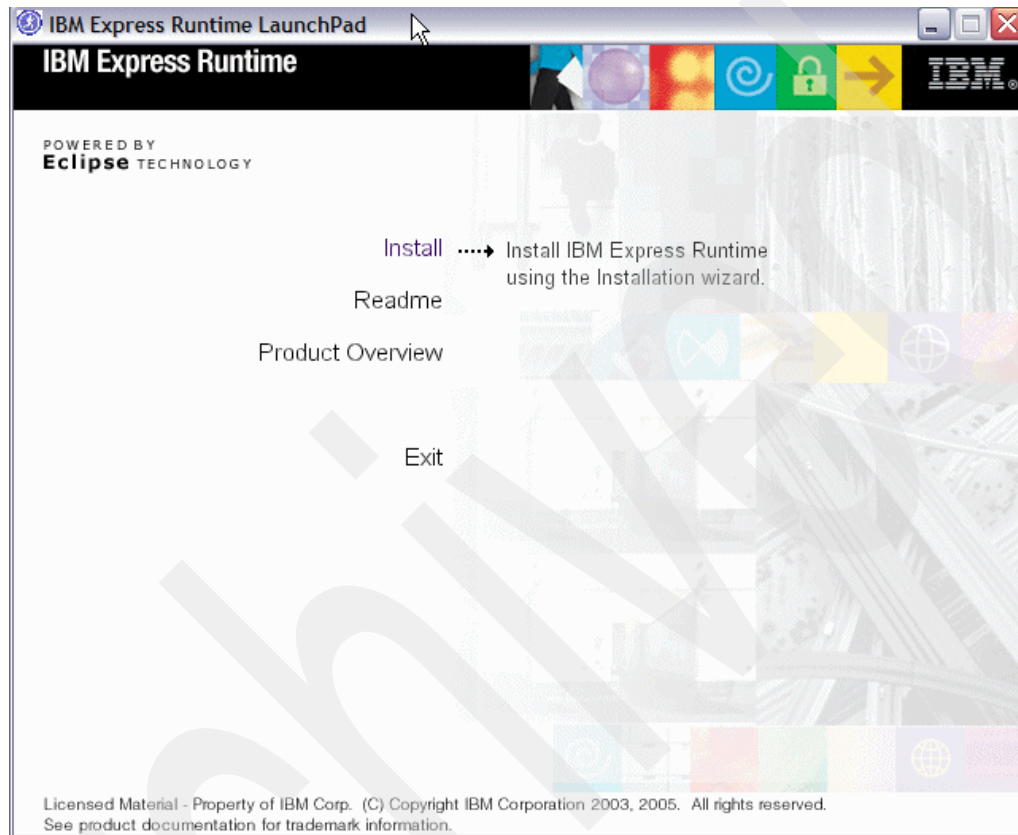


Figure 4-1 LaunchPad install panel for ERV2.1

- Regardless of the platform, the InstallShield Wizard starts as shown in Figure 4-2.

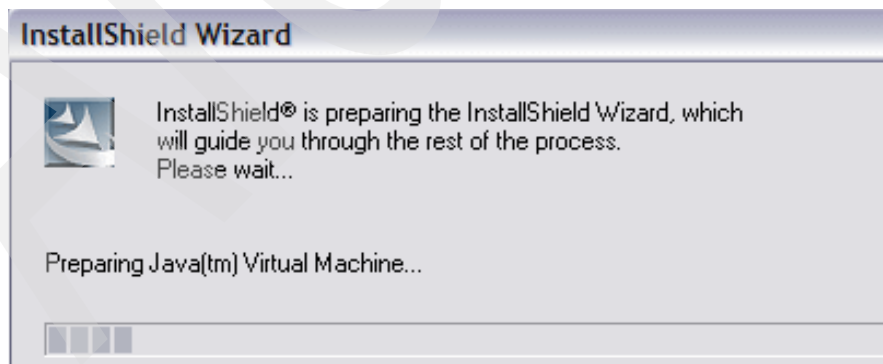


Figure 4-2 Express Runtime V2.1 InstallShield Wizard

4. Figure 4-3 notates the language selection for Express Runtime V2.1. Select a language and then click **OK**.



Figure 4-3 Express Runtime V2.1 language selection

5. Figure 4-4 shows the Express Runtime V2.1 Welcome page. Click **Next** to continue.



Figure 4-4 Express Runtime V2.1 Install Welcome

6. Figure 4-5 shows the Express Runtime V2.1 licensing agreement. Read and accept the terms and select the **Next** button to continue.

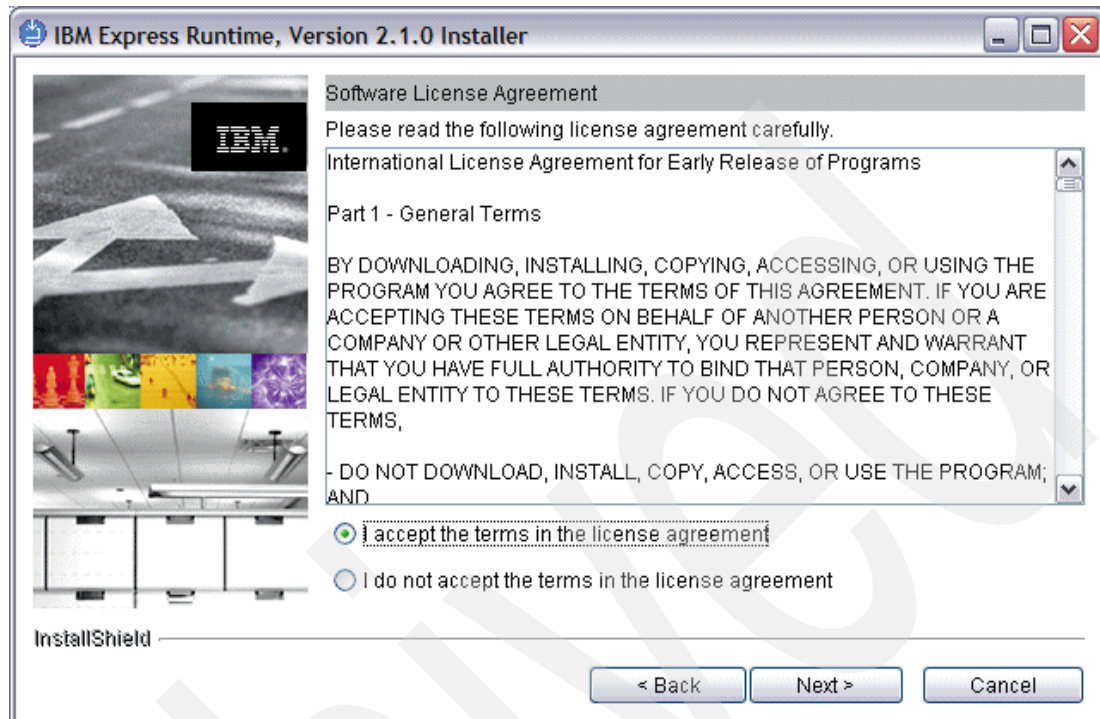


Figure 4-5 Express Runtime V2.1 Install License

Install IBM Express Runtime Option

Figure 4-6 shows the first option that was explained in 4.3, "Installation scenarios" on page 37. It installs the complete Express Runtime V2.1 package, including development tools, deployer tools, and middleware components.

7. Click **Next** to continue.

The second installation option is explained in "**Install only IBM Express Runtime middleware components**" on page 51.



Figure 4-6 Install option for whole Express Runtime V2.1 product

8. Figure 4-7 shows the default file location for the Express Runtime V2.1 install. Click **Next**.

Note: Users may change the default location of installed Express Runtime files.

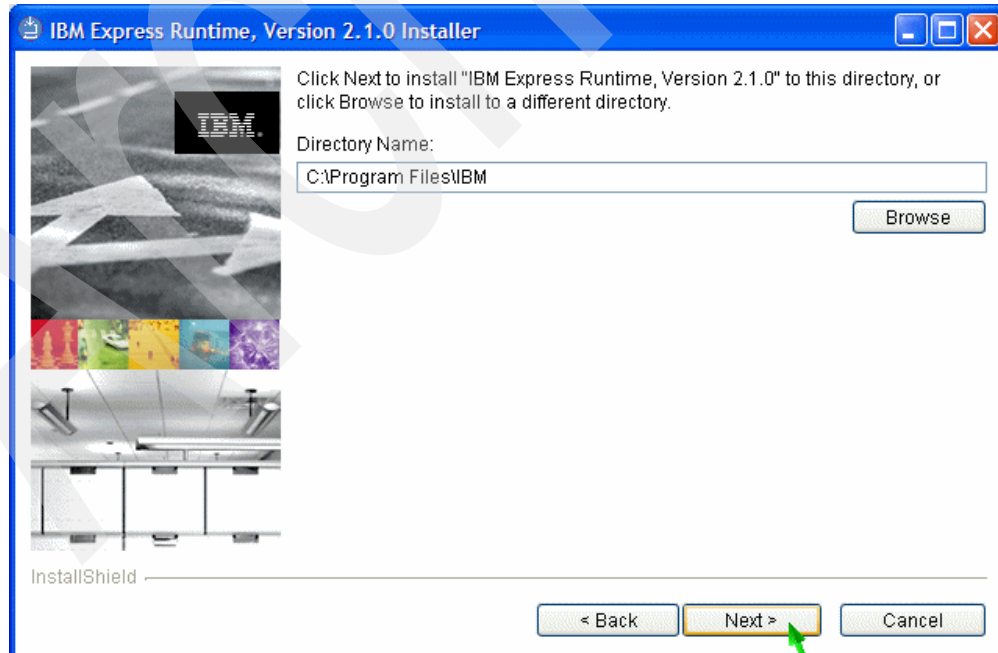


Figure 4-7 File location for installed files

9. If the directory has not been created from the action in Figure 4-7, you will get a warning message (see Figure 4-8) and the option to create the folder. Select **Yes** to continue.

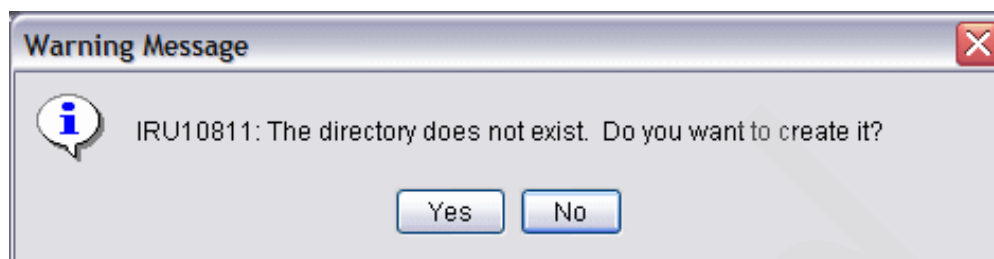


Figure 4-8 Express Runtime V2.1 Warning Message

Typical Installation

10. Figure 4-9 shows the two sub-options of Install IBM Express Runtime as outlined in "Installation scenarios" on page 37. Leave selection as Typical Installation and click **Next**.

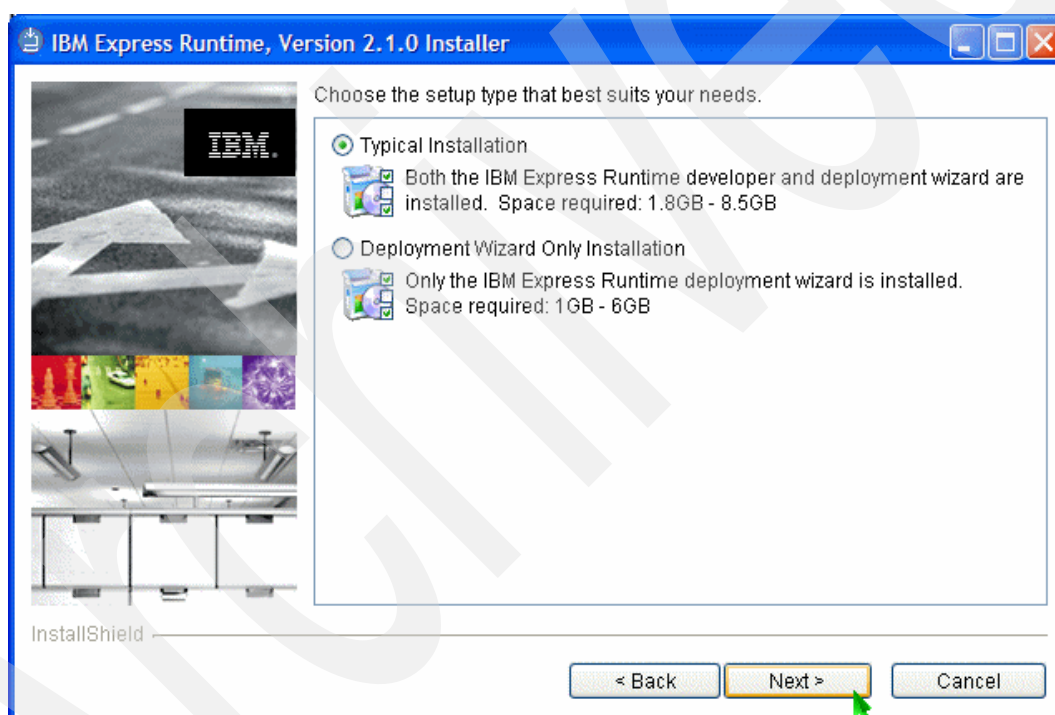


Figure 4-9 Setup Types

11. Figure 4-10 shows platform choices for middleware files needed for deployment of middleware. Choose the appropriate platforms (platforms that you plan to support with your solutions) and select **Next** to continue.

Note: Users can add platform specific middleware later.

Tip: You may find that the dialog is slow to respond if you select or deselect platforms. This is a known issue that will be resolved in a future release of Express Runtime. For this release, allow time for the dialog to respond to your clicks.

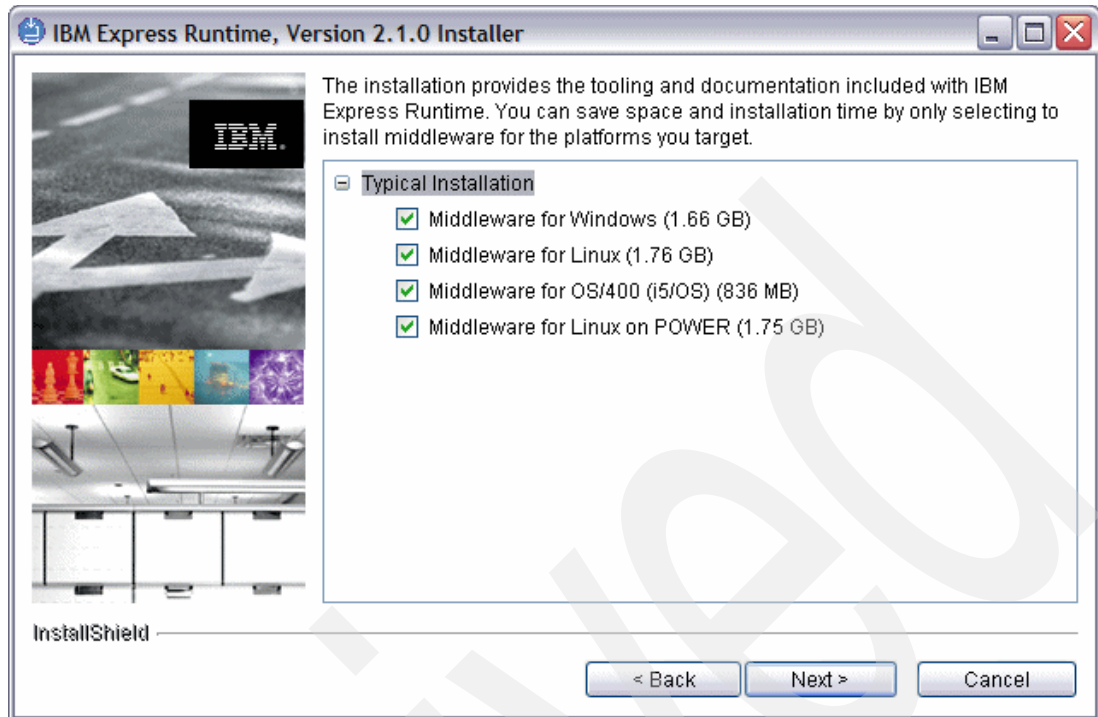


Figure 4-10 Typical installation Platforms for Deployment

12. Options shown in Figure 4-11 allow you to select the Eclipse based development environment. Users can either install Rational Web Developer or point to their own pre-existing Eclipse-based product. An existing Eclipse 3.0 product should be detected automatically, or you can manually select it by pointing to the directory that contains the eclipse.exe file. Select your option and click **Next** to continue.

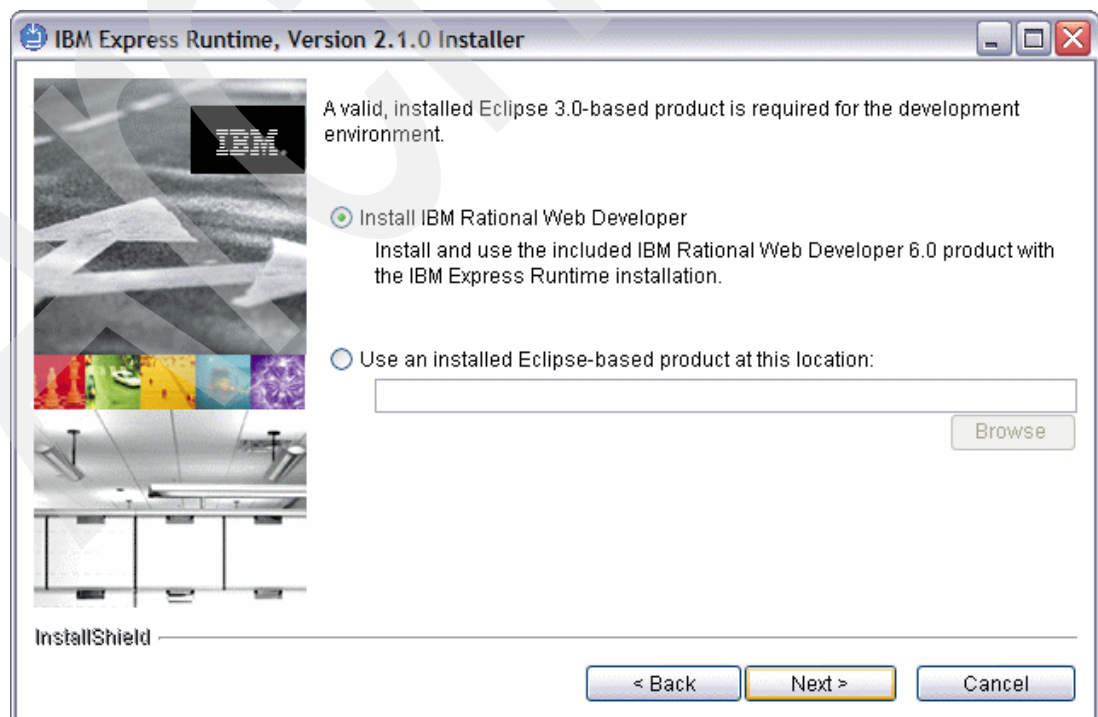


Figure 4-11 Typical Install Development Tool

13. Figure 4-12 provides a summary of the selected options and total size estimate before installing. Selecting the **Next** button will start the install.

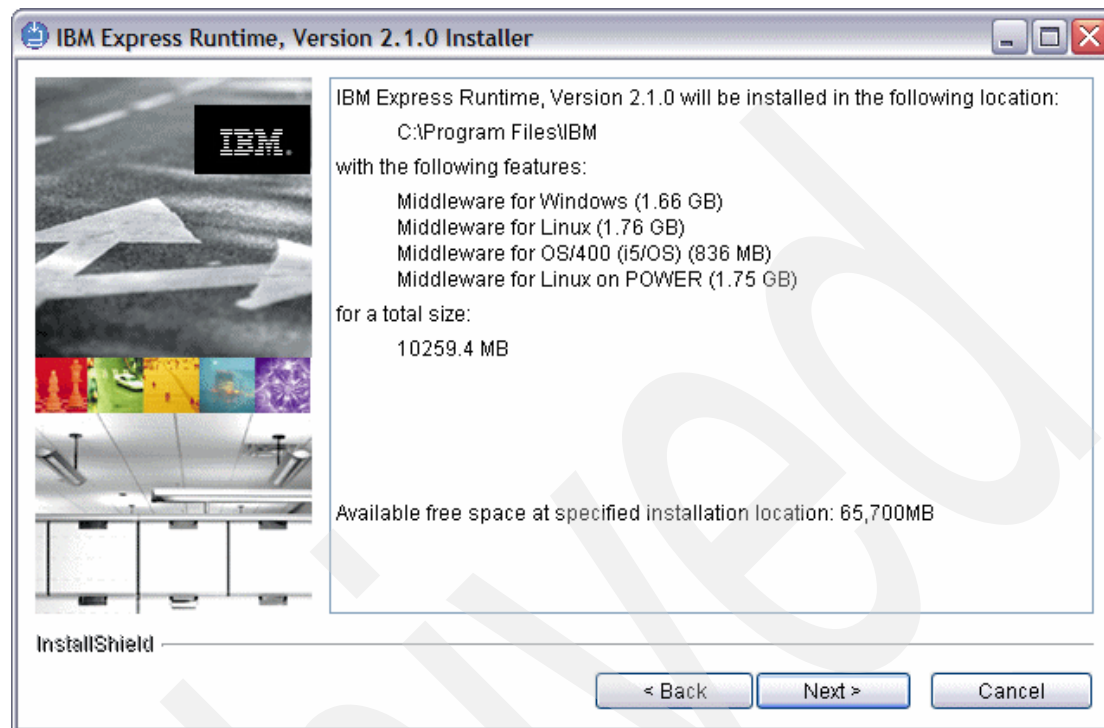


Figure 4-12 Typical installation Summary

14. Figure 4-13 shows the install bar for Express Runtime V2.1.0.

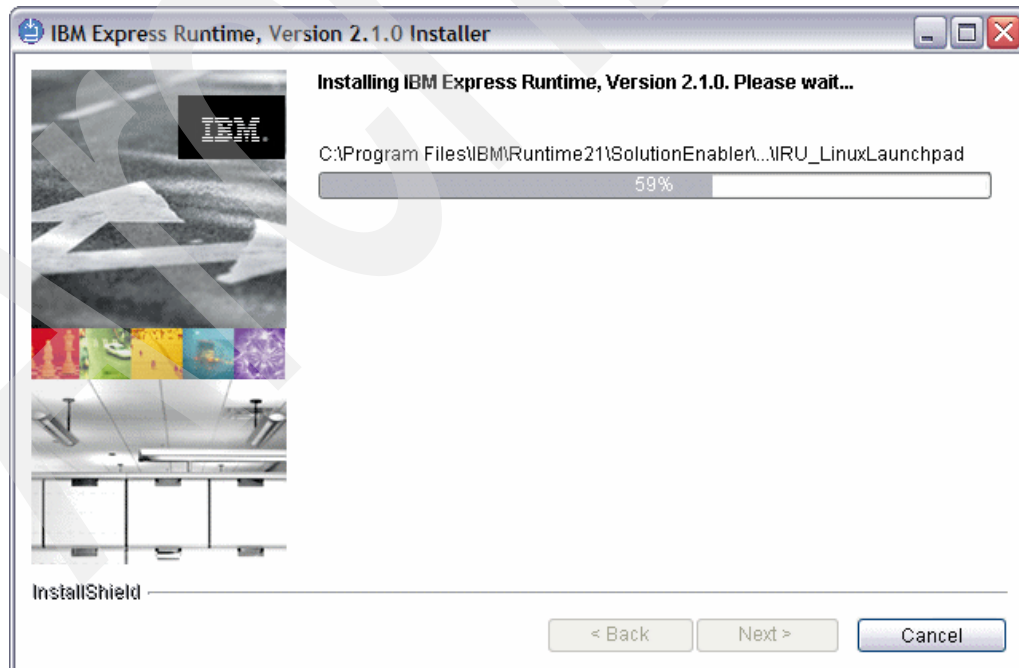


Figure 4-13 Typical installation status bar

15. Figure 4-14 shows the next part of the installation. The install bar description changes to notate the installation of Rational Web Developer 6.0. When the installation is complete, select the **Next** button.



Figure 4-14 Typical installation Rational Web Developer 6.0 status bar

16. When installation completes, click the **Finish** button to continue.

Deployment Wizard Only Installation

After proceeding from Figure 4-1 on page 38 through Figure 4-8 on page 43, the Deployment Wizard Only Installation option is available. This is the second sub-option for the Install IBM Express Runtime choice. Its install is limited to the deployment component. The deployment component includes:

- ▶ Deployment Wizard
- ▶ Sample Deployment Files including middleware, solutions and applications
- ▶ Documentation

1. Per Figure 4-15, choose the Deployment Wizard Only installation option. Click **Next** to continue.

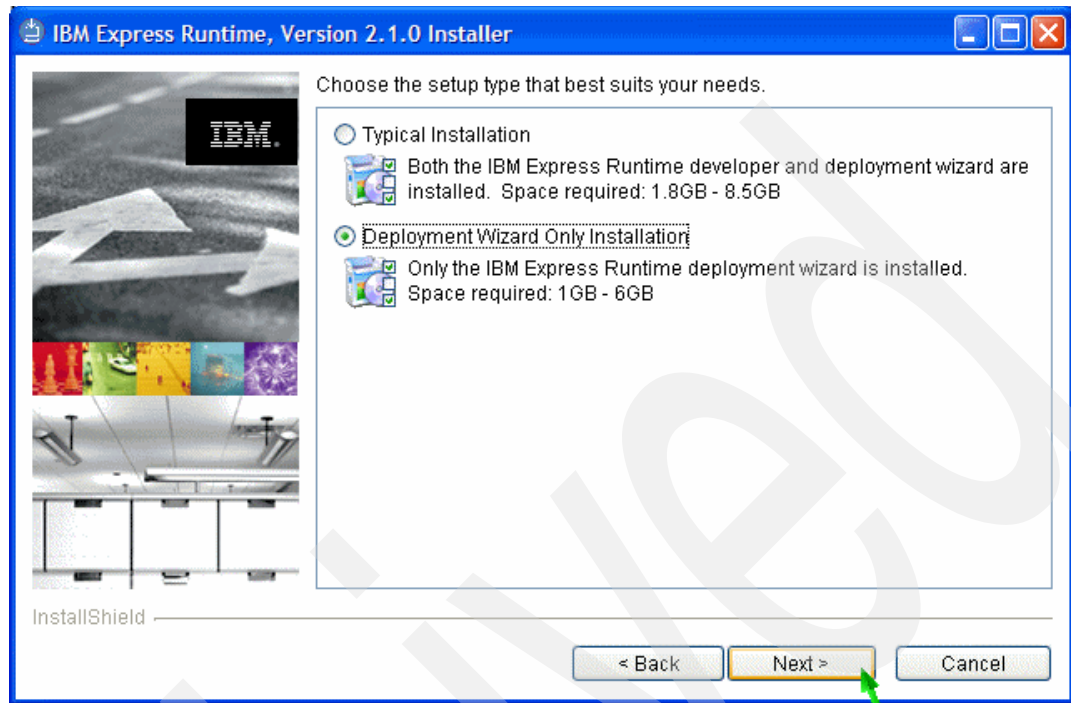


Figure 4-15 Deployment Wizard Only installation

2. Figure 4-16 allows the user to select which platforms are needed to deploy the middleware. In this example, Windows and Linux middleware can be deployed. Choose the appropriate platforms and select **Next**.

Note: Users can add platform specific middleware later.

Tip: You may find that the dialog is slow to respond if you select or deselect platforms. This is a known issue that will be resolved in a future release of Express Runtime. For this release, allow time for the dialog to respond to your clicks.

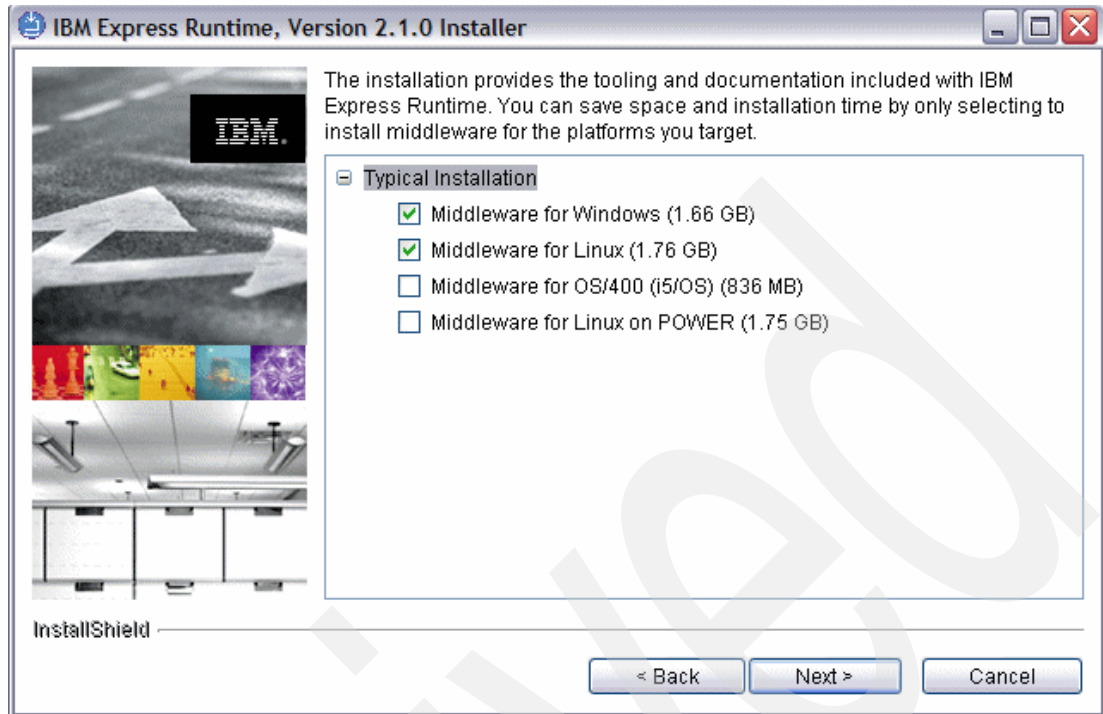


Figure 4-16 Platform selection for Deployment Wizard

3. Figure 4-17 shows a summary of the install. Click **Next** to continue.

The information included is:

- Where files are installed: C:\Program Files\IBM
- What is being installed: Windows and Linux Middleware
- What is the total file size: 3645.7 MB
- How much room is left at the specified location: 65,703 MB



Figure 4-17 Deployment Wizard Summary

4. Figure 4-18 shows the Deployment Wizard installation task bar for the Express Runtime V2.1 Deployment Wizard Only Installation. When it is complete, select **Next**.

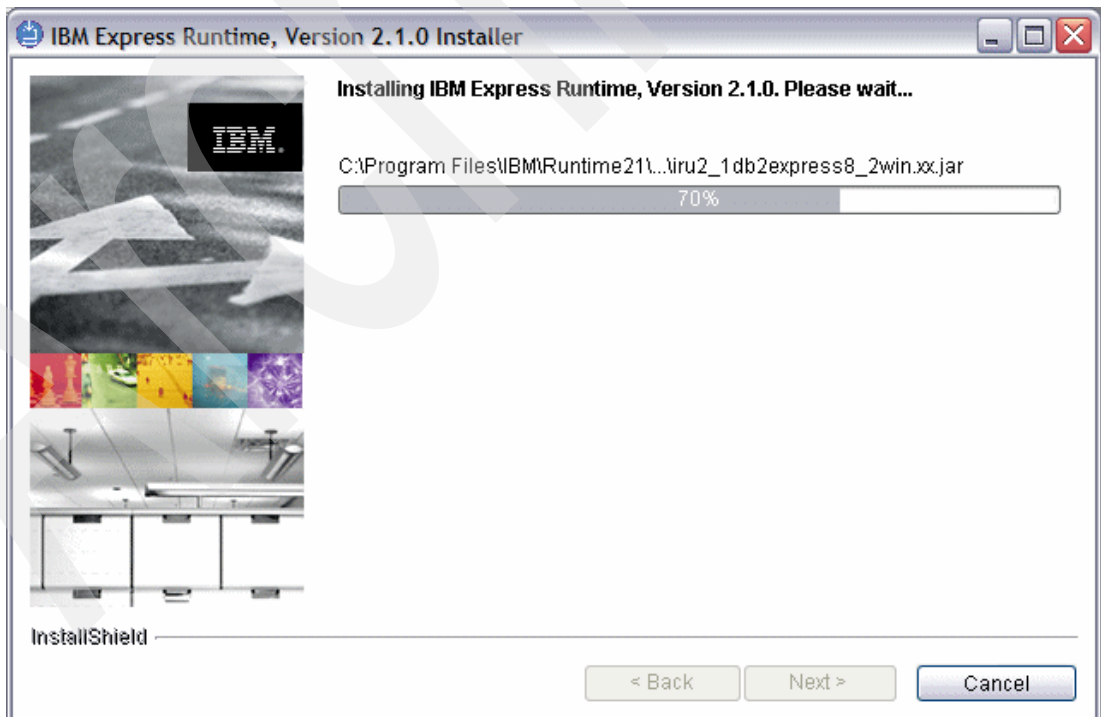


Figure 4-18 Deployment Wizard installation status bar

- Figure 4-19 shows the final panel for installing Deployer Wizard only. Select the **Finish** button.

Note: The check box can launch the First Steps documentation pane.

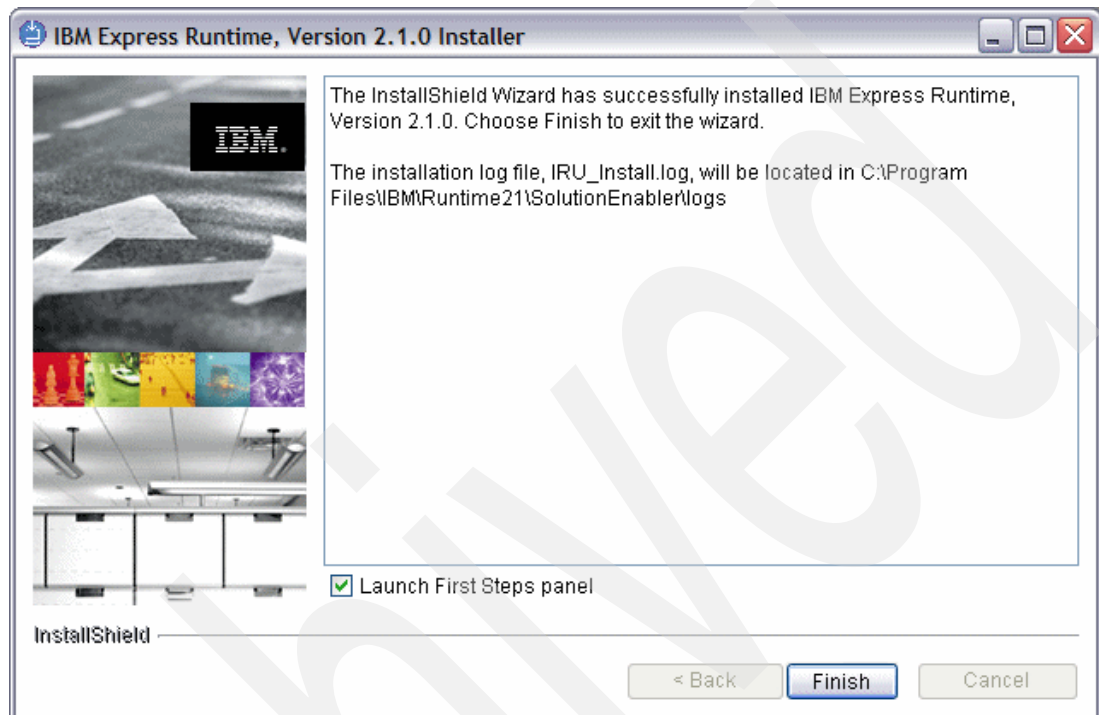


Figure 4-19 Finish Slide for Deployer only install

6. Figure 4-20 shows the Express Runtime V2.1 First Steps pane that was requested in Figure 4-19. This provides documentation and includes the following areas:
- ReadMe
 - Getting Started
 - Tutorials
 - Develop Sample Solution
 - Deploy Sample Solution
 - Express Runtime Developer
 - Deployment Wizard
 - Product Documentation
 - Exit

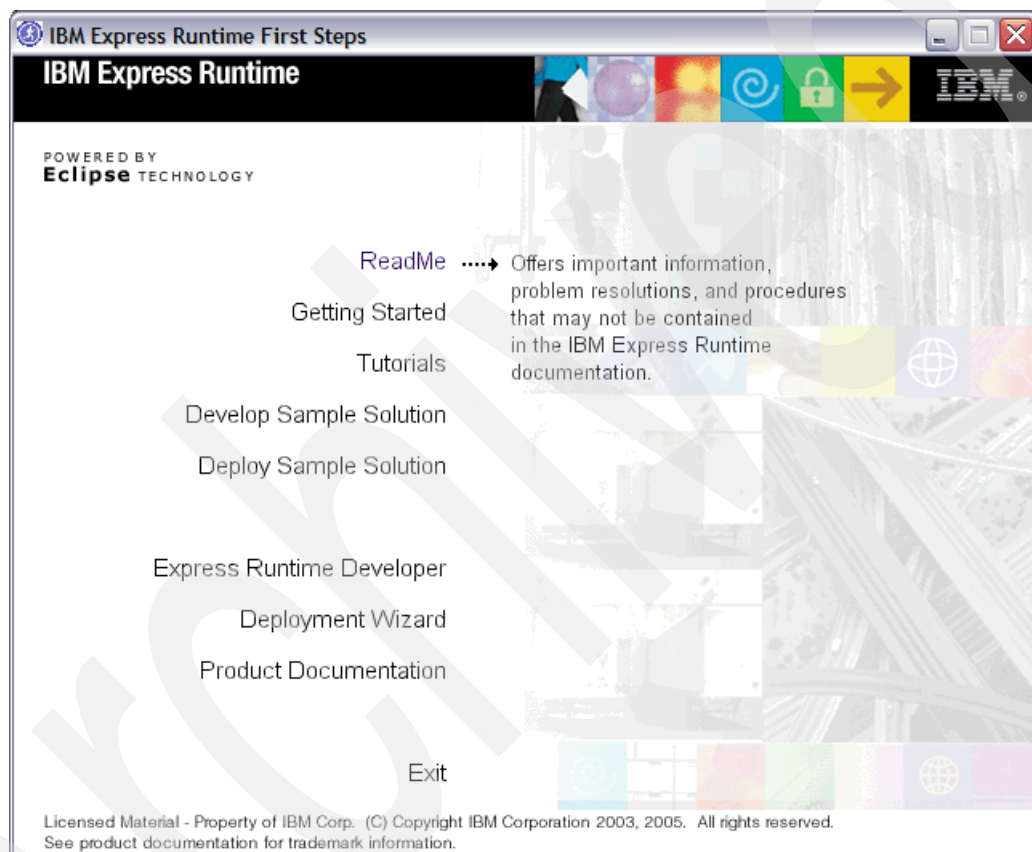


Figure 4-20 First Steps

Install only IBM Express Runtime middleware components

After the Figure 4-1 on page 38 to Figure 4-5 on page 41 section of the wizard, there is the option to Install only IBM Express Runtime middleware components as shown in Figure 4-21. It uses Solution Launcher to temporarily install Deployment Wizard, which then drives the deployment of the middleware components. When the deployment completes, Solution Launcher removes Deployment Wizard.

To deploy the middleware components, follow these steps:

1. Select **Install only IBM Express Runtime middleware components** and click **Next**.

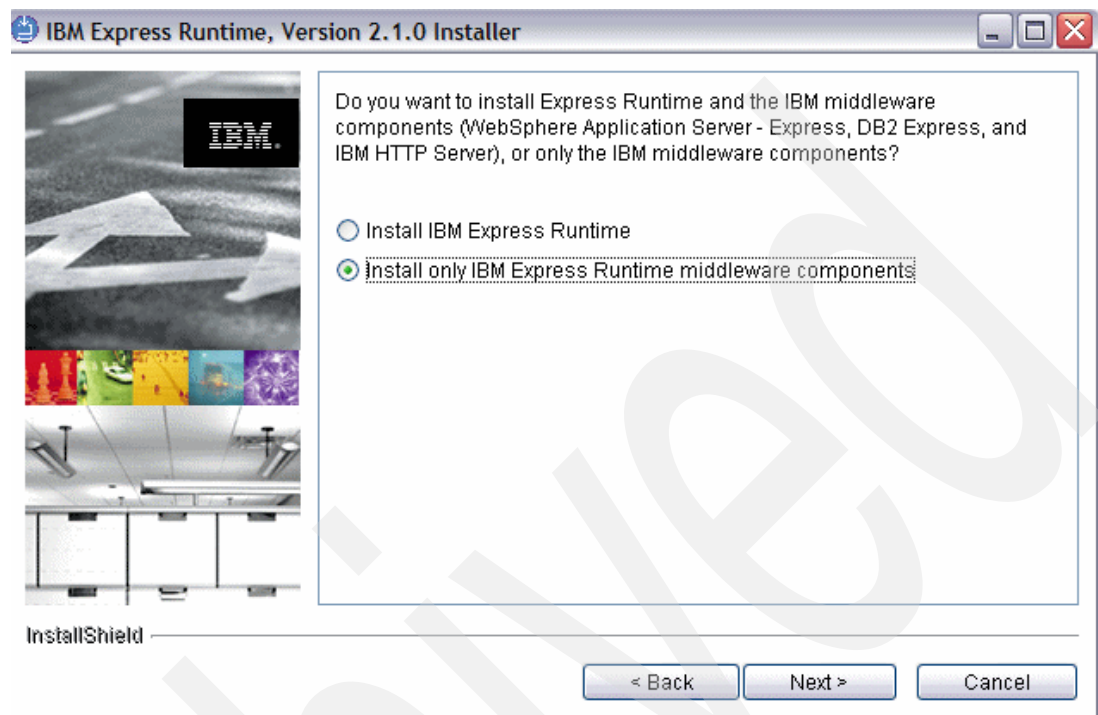


Figure 4-21 Install option for only Express Runtime V2.1 middleware components

2. Figure 4-22 shows the default folder directory for installation. After choosing the installation file location, select **Next**.

Note: Users can change the install file location for Express Runtime V2.1.



Figure 4-22 Express Runtime V2.1 install path

3. If the folder specified does not exist, the warning message shown in Figure 4-23 will appear along with the option to create it. Select **Yes** to continue.

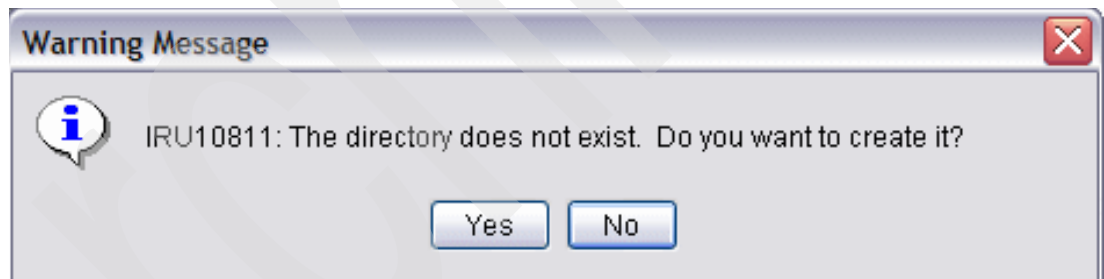


Figure 4-23 Express Runtime V2.1 Directory Warning Message

4. Figure 4-24 shows that the installation is ready to begin. Click **Next** to install.

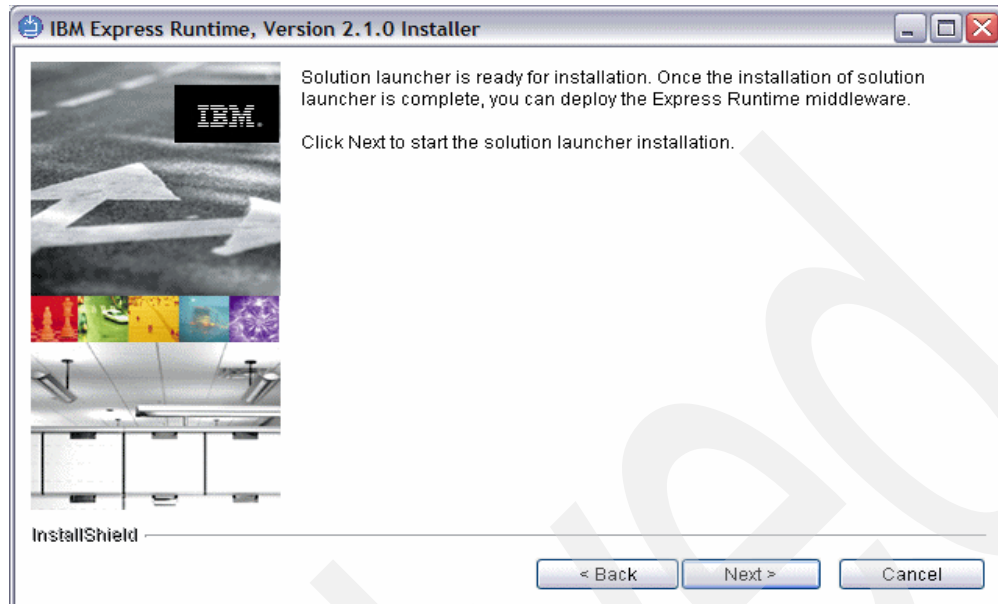


Figure 4-24 Express Runtime V2.1 Solution launcher installation panel

5. Figure 4-25 shows the task bar for installing the files required for solution launcher and Express Runtime middleware.

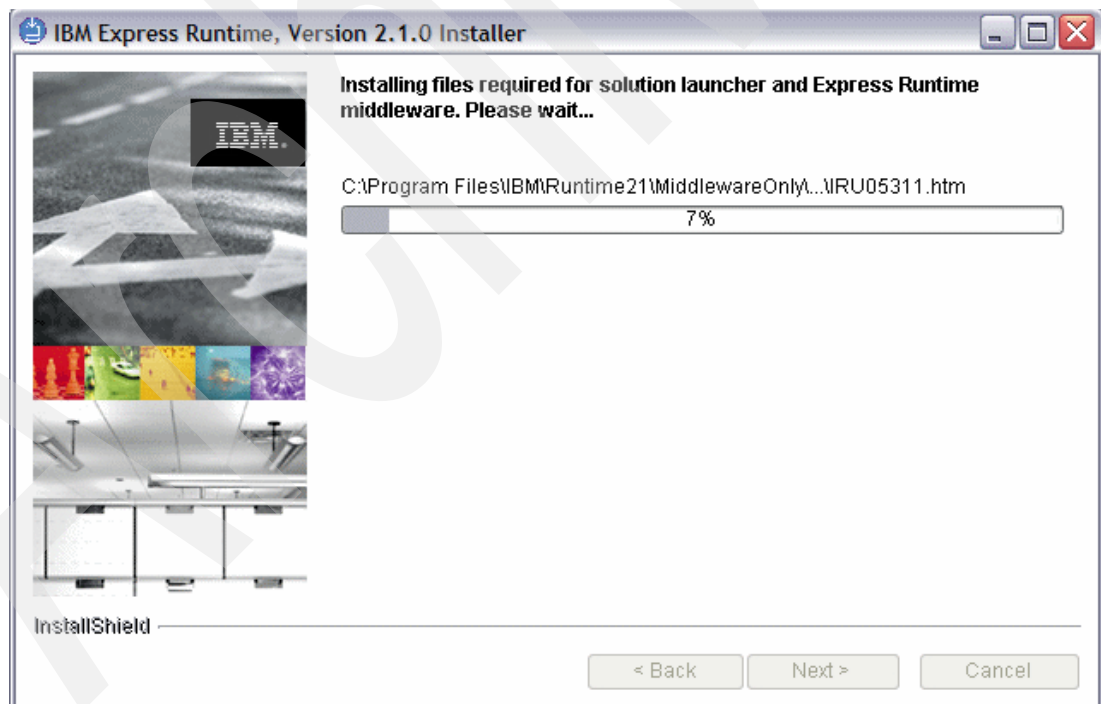


Figure 4-25 Express Runtime V2.1 Status Installer for Solution Launcher & middleware

6. Figure 4-26 shows the task bar for solution launcher installing the middleware files. These middleware files are used to install the middleware during deployment.

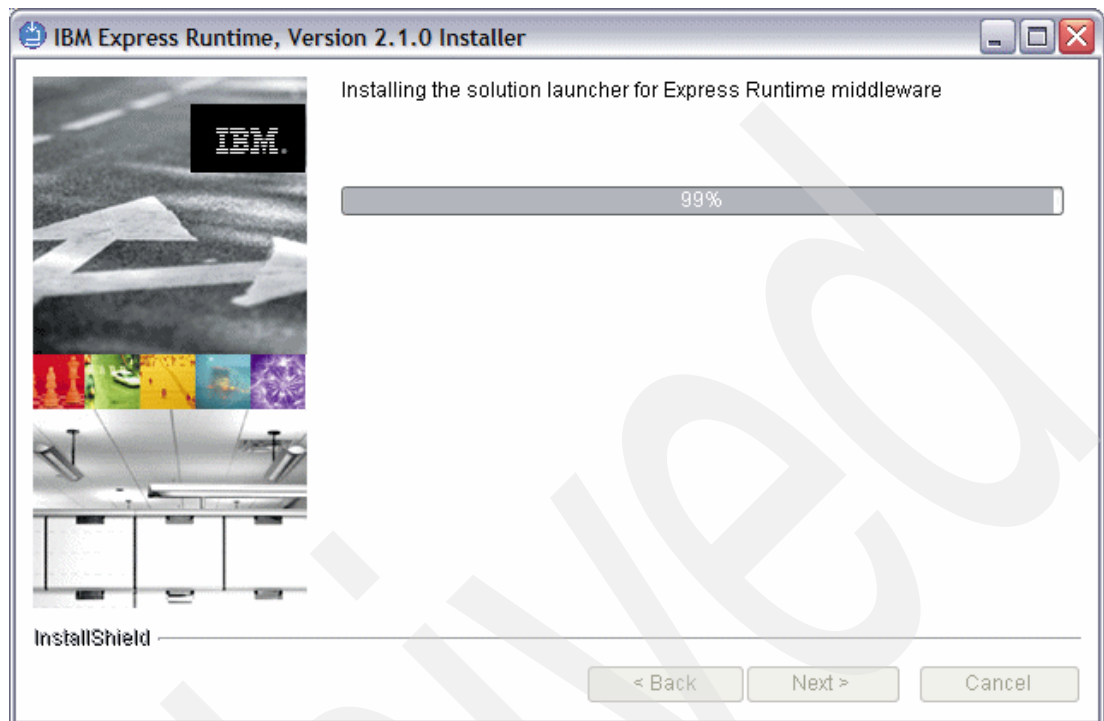


Figure 4-26 Express Runtime V2.1 Middleware Solution Launcher

7. Figure 4-27 shows the task bar for the uninstaller. When complete, select **Next**.



Figure 4-27 Creating Uninstaller

8. Figure 4-28 shows a successful installation of Solution Launcher. Click the **Finish** button to run Solution Launcher, which triggers Deployment Wizard to start.



Figure 4-28 Express Runtime V2.1 Middleware Setup Finish

9. Figure 4-29 shows Deployment Wizard. See Chapter 8, “Deploying a solution” on page 177, for information on deploying the middleware components.

Note: Closing the Deployment Wizard warns the user that Deployment Wizard will uninstall. Deploy your middleware first, or the necessary middleware files will be removed in the uninstall of the Deployment Wizard.

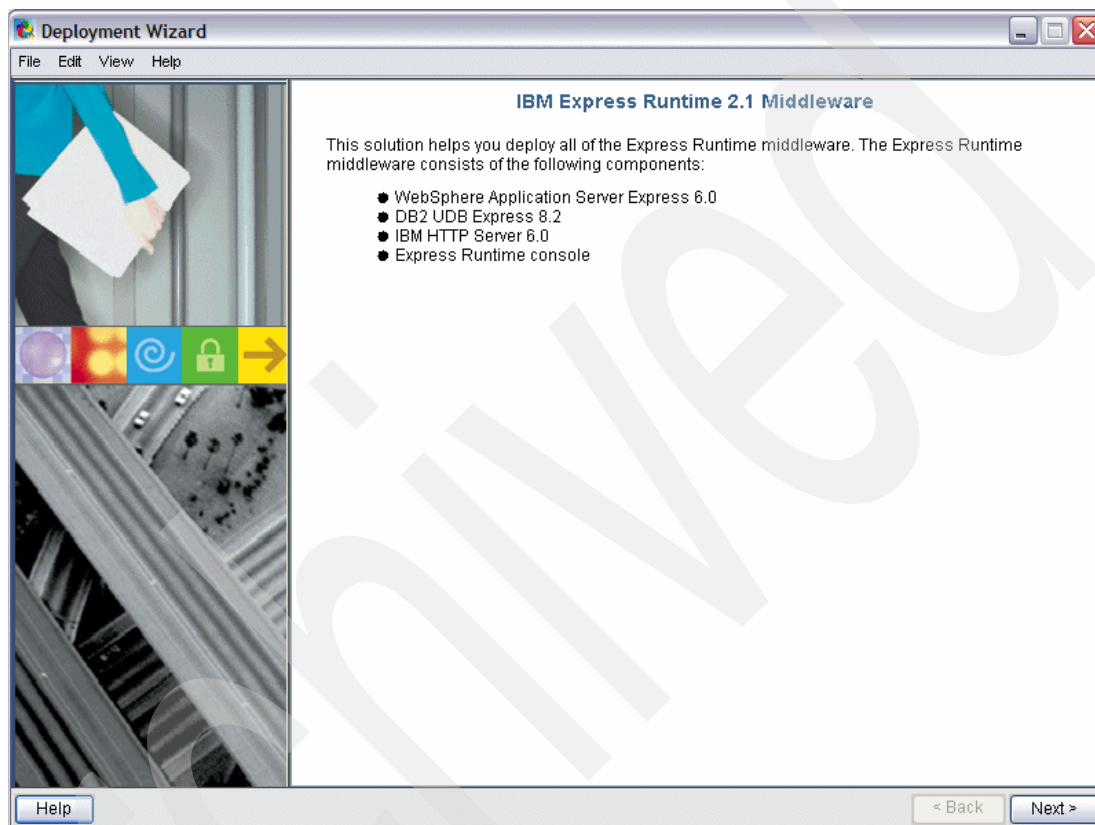


Figure 4-29 Express Runtime V2.1 Middleware Introduction

4.5 Exploring resulting Express Runtime V2.1 file directory and software location

Depending on the platform, you will find the installed product files in the various locations.

4.5.1 Windows

As shown in the install process, the default install file location is C:\Program Files\IBM\Runtime21. When you select **Start** → **Programs**, you see IBM Express Runtime V2.1 and IBM Rational.

4.5.2 Linux and Linux on IBM Power

As demonstrated in Example 4-2, Express Runtime V2.1 files are stored in /opt/IBM/Runtime21/.

Example 4-2 Express Runtime V2.1 files

```
[root@relinux Runtime21]# cd /opt/IBM/Runtime21/
[root@relinux Runtime21]# ls -l
total 36
drwxrwxrwx  2 root    root      4096 Jan 13 09:34 firststeps
-rwxrwxrwx  1 root    root      236 Jan 13 09:34 First_Steps_shortcut.sh
drwxrwxrwx  2 root    root      4096 Jan 13 09:31 graphics
drwxrwxrwx  5 root    root      4096 Jan 13 09:31 info
-rwxrwxrwx  1 root    root      100 Jan  8 20:57 L5724F71010100.sys
drwxrwxrwx  4 root    root      4096 Jan 13 09:31 license
-rwxrwxrwx  1 root    root     2167 Jan  8 20:54 RWD_linux.iss
drwxrwxrwx 16 root    root      4096 Jan 13 12:44 SolutionEnabler
drwxrwxrwx  2 root    root      4096 Jan 13 09:49 _uninst
[root@relinux Runtime21]#
```

4.6 Uninstalling the product

The Express Runtime V2.1 installation process creates the uninstaller. It is located in the folder under the product install directory and called `_uninst` (see Example 4-2 for Linux and Figure 4-30 for Windows).

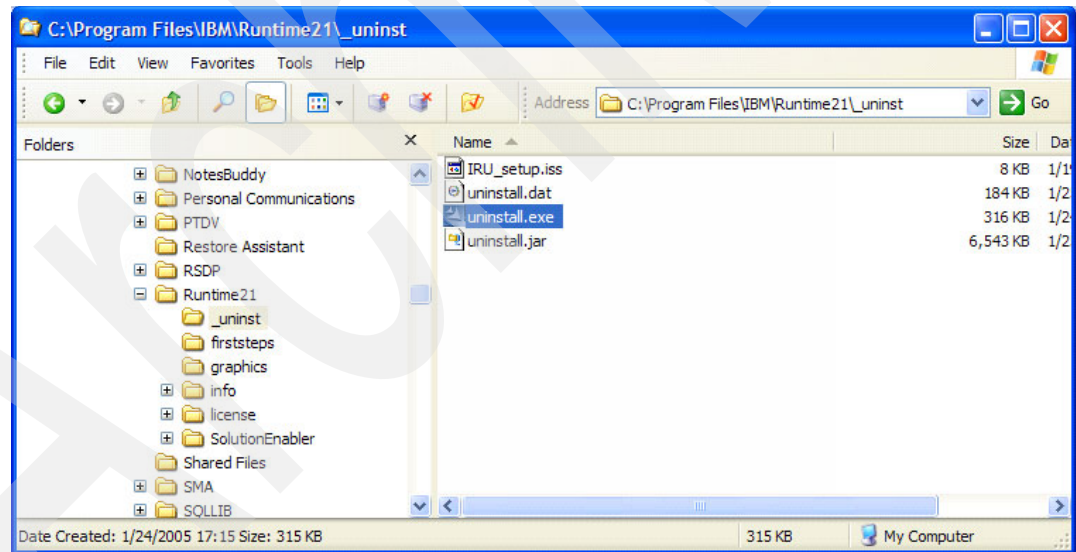


Figure 4-30 The content of the _uninst folder

Run:

- ▶ uninstall.exe on Windows
- ▶ uninstall in Linux.

This starts the InstallShield Wizard for uninstalling Express Runtime V2.1.

1. Figure 4-31 shows the Uninstaller Welcome page. Click the **Next** button.



Figure 4-31 Express Runtime V2.1 Uninstaller Welcome page

2. Figure 4-32 shows the file location of Express Runtime. Click the **Next** button.

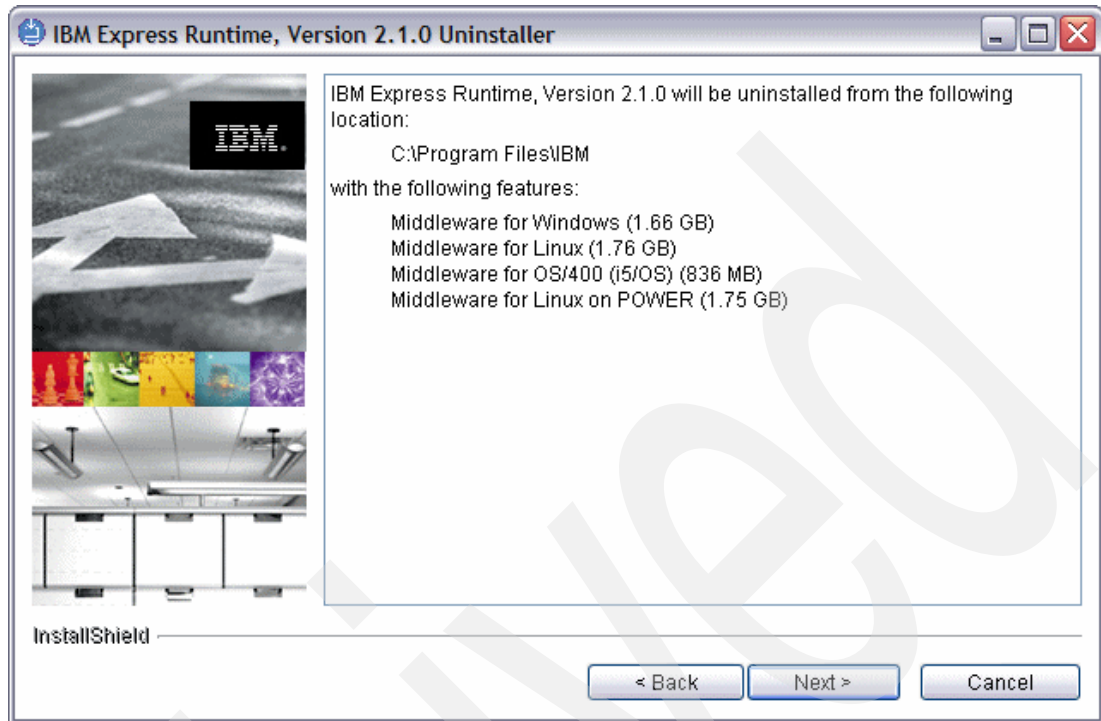


Figure 4-32 Express Runtime V2.1 Uninstaller File Location

Note: This panel shows the directory path for product files. Often, the wizard does not remove all the files. Looking in this path helps the user to know where to go to remove residual files.

3. Figure 4-33 shows the status bar. When it is done, select the **Next** button.

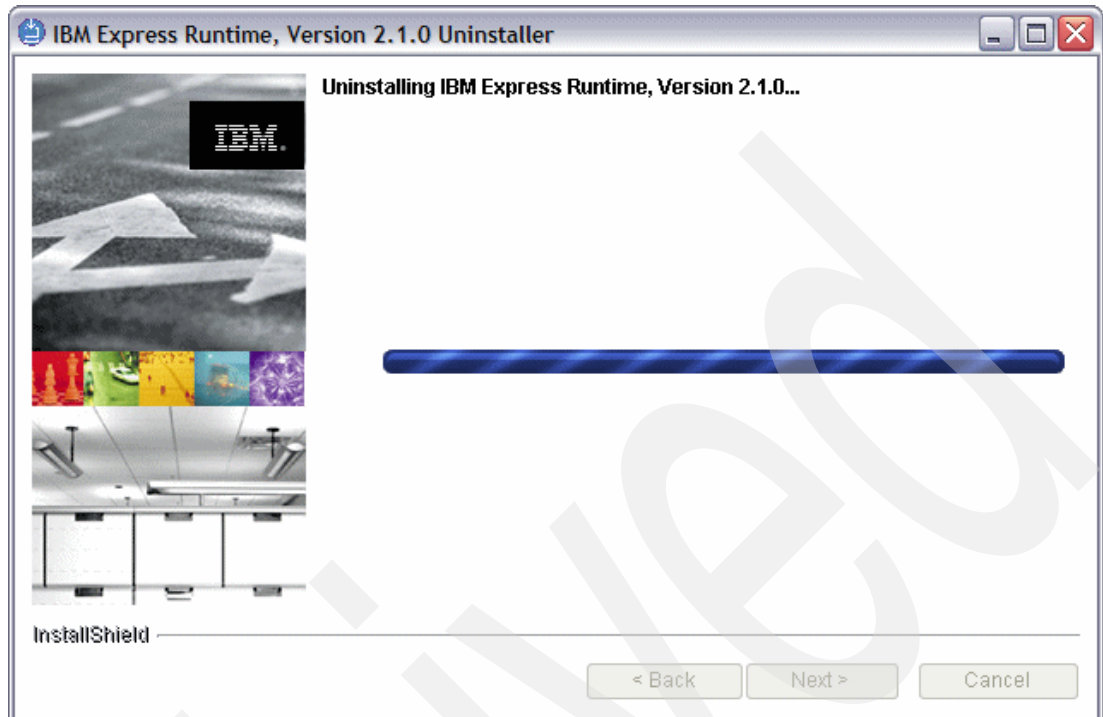


Figure 4-33 Express Runtime V2.1 Uninstaller Status Bar

4. Figure 4-34 shows the final Finish panel. Click the **Finish** button and you are done with the wizard section.

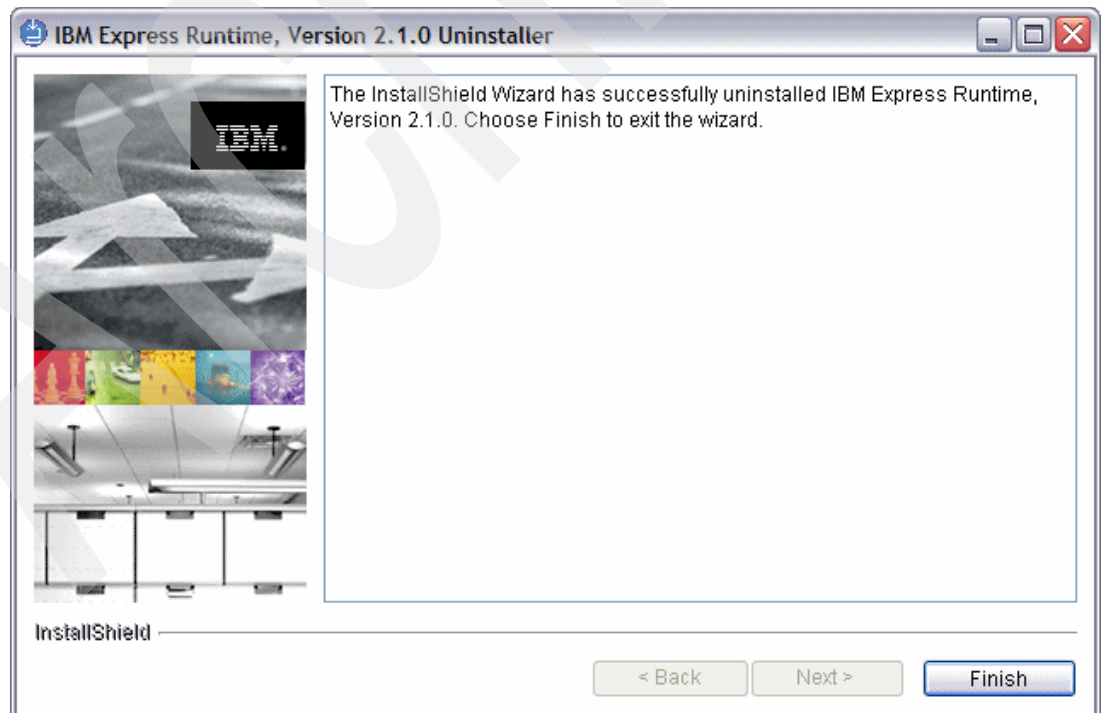


Figure 4-34 Express Runtime V2.1 Uninstaller Finish panel

4.6.1 Keep in mind when you uninstall

Keep in mind the following points when you uninstall the product:

1. Some files and folders under the Express Runtime install directory are not deleted by the uninstall wizard. You have to delete them manually.
2. Rational Web Developer or Eclipse-based development tool is not uninstalled by the wizard.
3. None of the deployed middleware components and/or applications are removed. You have to do it individually using the appropriate uninstall method for each component.
4. You can run the silent uninstall. You need to use the IRU_setup.iss file in the _uninst directory to specify the uninstall options. For Windows, for example, you need to run the following two commands at the command prompt:

```
cd "<ER_INSTALL_DIR>\_uninst"  
uninstall.exe" -options IRU_setup.iss
```


What's a wrapper?

This chapter provides an introduction to the concept of an application and solution wrappers. If you want to create a custom solution that contains your application, then creating a wrapper is the main development activity that you must perform. This chapter describes the process for creating wrappers using the Express Runtime Developer tool.

5.1 Exploring a wrapper

As used in Express Runtime V2.1, wrappers can be thought of as the “workflow” or “controller” elements that are responsible for controlling the flow of execution for deploying a component.

There are two types of wrappers:

- ▶ Application wrapper
- ▶ Solution wrapper

5.1.1 Application wrapper

In the context of Express Runtime V2.1, an *application* is best thought of as a component of an overall solution. This can include a middleware component as well as an actual application in the more generalized sense. For example, one component of a solution might be WebSphere Application Server Express. Although this is a piece of middleware, it is still an *application* in the terminology used by Express Runtime.

In order to integrate an application into a solution, a developer has to create a custom wrapper for this application.

Important: Make sure you understand the term *application* as it applies to Express Runtime V2.1. An *application* is a building block of an Express Runtime solution. It can be a J2EE application or a middleware component.

Each application wrapper contains of several XML files, user programs, and optionally other artifacts, such as scripts, GIF files, and so on. These components are all contained within an application project that you create using the Express Runtime Developer tool.

The XML files define the way the application is installed and configured on the target system. One of these XML files called *application.xml*. This is the most important file. It contains all information about the user programs, application files, variables, and libraries.

User programs run during the application deployment to perform the installation, as well as pre-installation or post-installation operations. User programs can be used to:

- ▶ Drive the installation of IBM middleware components as well as partners’ applications.
- ▶ Execute the DB2 scripts to create tables required for application and populate the table with data.
- ▶ Configure middleware component, such as WebSphere Application Server Express, using wsadmin scripts written in Jython or JACL.

User programs can use one of the several options, such as a return code, to indicate the success or failure. A user program can be one of the following types:

- ▶ *Java program*.
- ▶ *Custom program*. It can be a custom script, executable or system command.
- ▶ *InstallShield executable*.

The typical choice for a user program type is a Java program.

After the application wrapper development completes, the application is packaged together with the middleware components required to support the application. The component that allows you to do this is called a *solution wrapper*.

5.1.2 Solution wrapper

A solution consists of one or more applications. In order to achieve the integration of multiple applications in a solution, you need to develop a *solution wrapper*, which defines applications that will be installed and configured, their order, and variables shared by applications. This information is defined in a special XML file called *solution.sxml*.

The solution wrapper doesn't require any programming activities.

5.1.3 Conclusion

To conclude this discussion, look at Figure 5-1 for a conceptual view of the entire solution. Notice how application and solution wrappers fit into this picture.

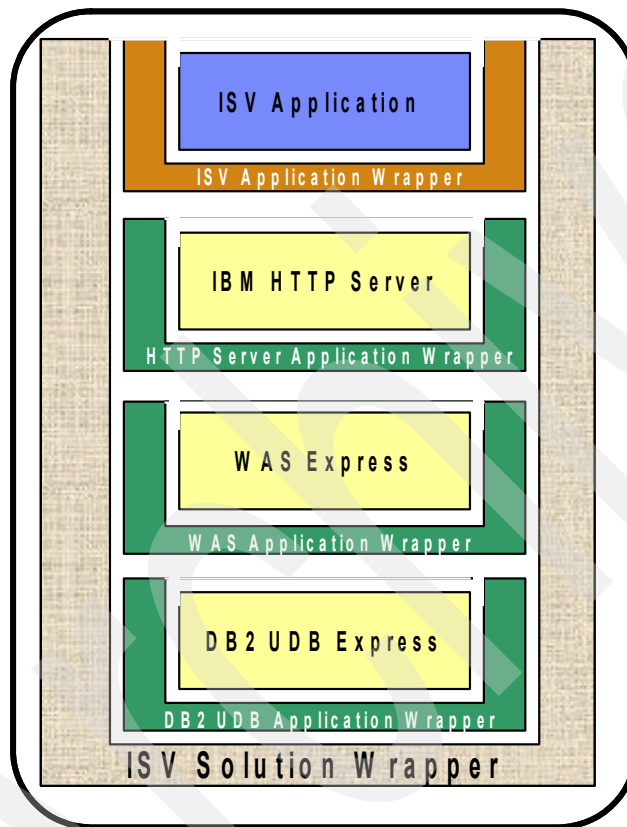


Figure 5-1 The conceptual view of a complete solution

5.2 Developing a wrapper

In this section we describe the high-level steps in developing a wrapper. This information lays down a basis for understanding the real-life examples described in Chapter 6, "Developing a wrapper" on page 91.

5.2.1 Developing an application wrapper

Before you start developing an application wrapper, make sure you test your application for functional errors. Only when your testing is successful, should you start developing a wrapper.

Figure 5-2 provides a logic view of the development process.

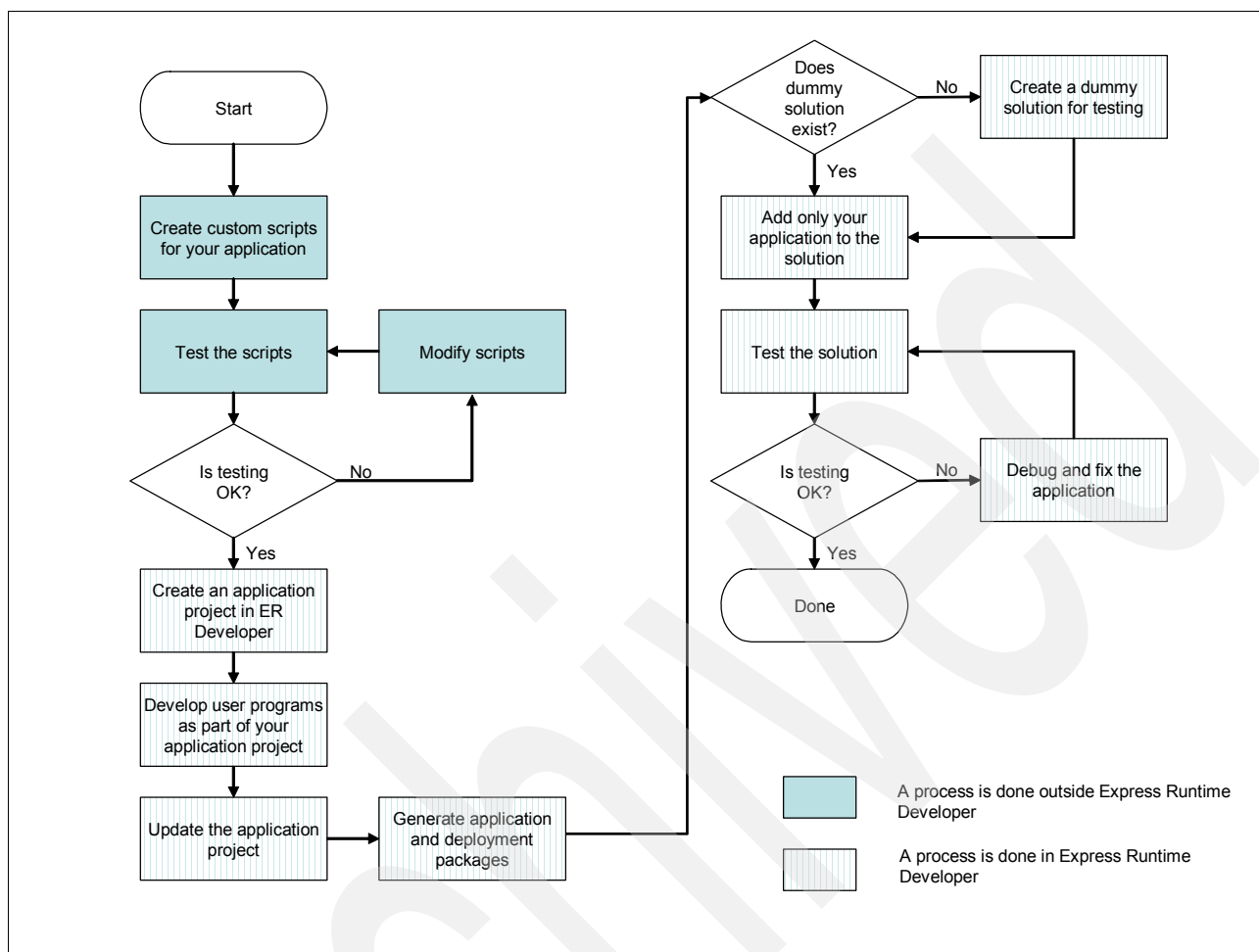


Figure 5-2 The overall process for developing a wrapper

Let us now discuss the most important steps in this process in more detail.

Step: Create custom scripts for your application

In order to prepare your application for Express Runtime V2.1, you need to automate the process of installing and configuring your application. For example, if you have a J2EE application, you need to develop the wsadmin script to install this application into WebSphere Application Server. In addition, you need to create the DB2 scripts to create and populate the database.

Step: Test the scripts

In this step you test your scripts. It is normally a good practice to test your application as you develop it. In this case, it is much easier to find an error related to the scripts, then to do this when you test the entire solution.

In our example with the J2EE application, you need to have all the required infrastructure components installed (IBM WebSphere® Application Server - Express Version 6, DB2 UDB, IBM HTTP Server, and so on) before testing the scripts.

Step: Create an application project in Express Runtime Developer

The development of the application wrapper in Express Runtime Developer is done under the Application Project. This is a predefined project template that lays down most of the plumbing for the application wrapper development. A special wizard guides you through the creation process.

When the wizard exits, you should have a skeleton of your project created. One of the most important parts of this project is a special file - *application.xml*. This is the required file.

Step: Develop user programs as part of your application project

You can create several different types of the user programs. A typical user program is written in Java. Rational Web Developer, which is included with Express Runtime V2.1, contains a full IDE for Java development.

Express Runtime Developer comes with the support APIs (see Section 5.3, “Support Framework API” on page 86), which simplify and streamline the development of the user programs. These APIs support all four platforms.

The only required user program is *Main Program*. However, you have three additional types of user programs:

- ▶ Predeployment checker
- ▶ Entry program
- ▶ Exit program

Predeployment checker

Before an entry, exit, or main program is run, the application's deployment package must be transferred to the target computer. A predeployment checker runs prior to the application entry, main, and exit programs, and prior to the deployment package being transferred to the target computer. So, it prevents transferring the solution files to the target system if the pre-conditions are not met.

A predeployment checker could ensure that the target computer met the following requirements:

- ▶ Adequate disk space is available on the target computer.
- ▶ System requirements are sufficient.
- ▶ There is no conflicting software.
- ▶ No previous version of the application exists.

When the predeployment checker exits, it returns one of the following return codes:

- ▶ 0 indicates that the application deployment should continue.
- ▶ 1 indicates that the application deployment should be skipped because the application is already installed or the application does not apply. The deployment wizard continues to the next application.
- ▶ A negative value indicates that the application cannot be deployed because of a fatal error, ending the deployment.

Entry program

Entry programs are typically used to perform pre-configuration or setup required before the main installation program runs.

Main program

The main user program is the installation program that is run on the target computer. Any type of installation program is acceptable as a main program, provided it can be invoked from a command line and run silently.

Exit program

Exit programs are typically used for post-configuration or cleanup required following the product installation. Exit programs can also be used to determine if a main installation program installed successfully.

Note: The return code values for all predeployment checkers, entry, exit, and main programs running on Linux or OS/400 (i5/OS) platforms must be between -128 and 127.

Step: Update the application project

When you finish the user program development, you need to update the application.xml file with the details about:

- ▶ User programs
- ▶ Application files, such as a EAR file
- ▶ Variables
- ▶ Libraries, such as a JAR file to support your application

Express Runtime Developer comes with *Application Wrapper Editor*. This is a special editor that display the parameters in the application.xml file in easy to read/edit view.

Figure 5-3 shows the Welcome tab of Application Wrapper Editor.

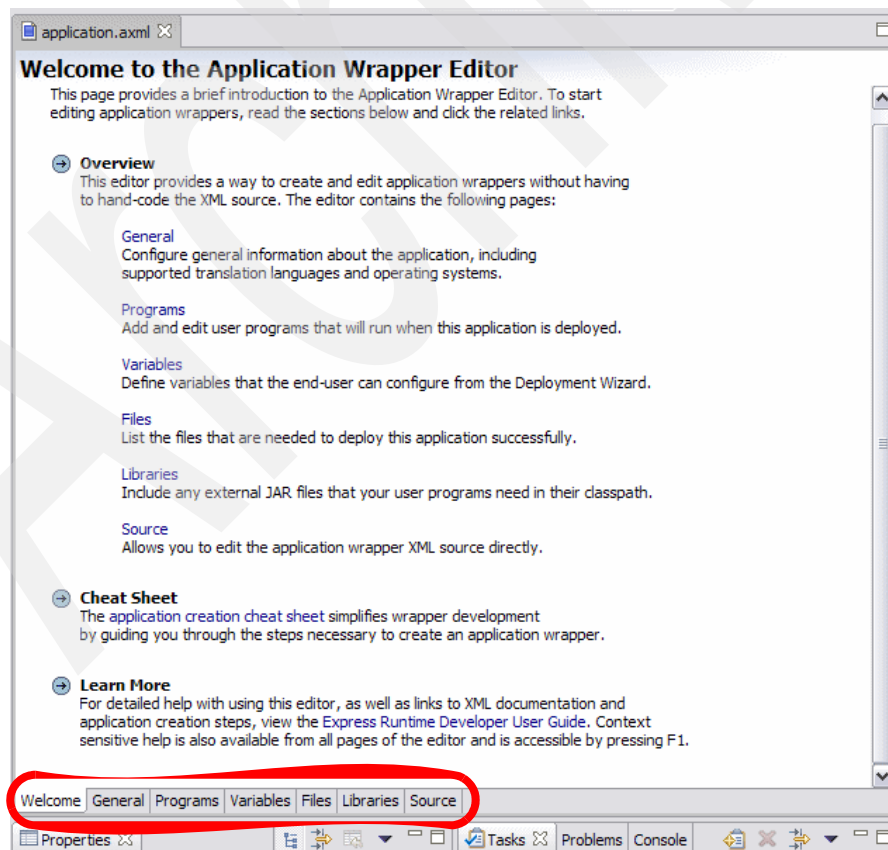


Figure 5-3 Welcome tab in Application Wrapper Editor

There are several tabs at the bottom of the editor's window that allow you to easily switch between different groups of the parameters.

The help facility in Express Runtime developer allows you to open a cheat sheet for the application wrapper development. Select **Help** → **Cheat Sheets**. In the pop-up windows click **Express Runtime** in the left pane and then **Creating an application wrapper** in the right pane. The help window shown in Figure 5-4 opens.

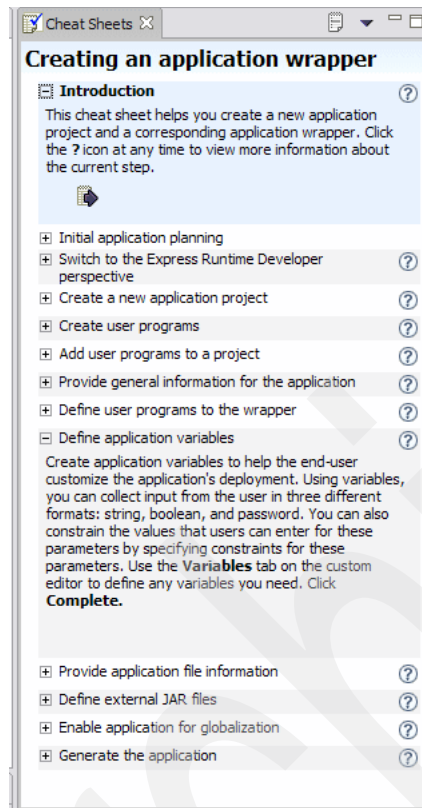


Figure 5-4 Cheat Sheets for Creating a application wrapper

You can expand each step to see more information about that step.

General tab

The General tab is shown in Figure 5-5. The fields displayed on the General tab are populated with the information collected by the application project wizard. However, you can modify them as needed.

The screenshot shows the 'General Application Information' dialog box. It is divided into four main sections: 'Basic Application Configuration', 'Supported Translation Languages', 'Advanced Application Configuration', and 'Supported Operating Systems'. The 'Basic Application Configuration' section includes fields for *ID (IRU2_1SampleWin), *Application name (%name), *Version (2.0), Installation time (5 minutes), Provider name (%providerName), and License text. The 'Supported Translation Languages' section has a 'Default' dropdown set to 'English' and a list of languages with checkboxes: Chinese, Chinese (Taiwan), English, French, German, Italian, Japanese, Korean, Portuguese (Brazil), and Spanish. The 'Advanced Application Configuration' section includes 'Deployment package protected' (checked), 'Deployment package name' (iru2_1sample), and 'Configuration instructions' (%configureText). The 'Supported Operating Systems' section has a list of operating systems with checkboxes: Linux, Linux on POWER, OS/400 (i5/OS), and Windows. The dialog box has a tabbed interface at the bottom with tabs for Welcome, General, Programs, Variables, Files, Libraries, and Source. The 'General' tab is currently selected.

Figure 5-5 General tab for Application Wrapper in GUI

Provide any additional information for the application. For most of the fields on the General tab, you can either enter text directly, or provide a translation key that is used to look up translated values in language-specific translation files. To use a translation key, enter a key name and prepend with the percentage sign (%).

For more information on supported languages see Section 5.4, “Support for other languages” on page 87.

Programs tab

The Programs tab in Figure 5-6 is used to define the user programs. The only required attributes for any user program are the program type and the program name. Other attributes that might be needed for your user programs include, but are not limited to, the log file name, response file name and any argument you want to pass to the user programs.

The screenshot displays the 'Application Wrapper Editor' window with the 'Programs' tab selected. The window title is 'application.xml'. The main section is titled 'User Programs Information' and contains a list of user programs. Below this, there are several configuration sections: 'Basic Program Configuration', 'Program Success Type', 'Custom Program Options', 'Java Program Options', 'Advanced Program Configuration', and 'Program Arguments'. The 'Basic Program Configuration' section shows 'Program type' as 'Java program' and 'Program' as 'com.ibm.iru_samplewin.SampleWinPDC'. The 'Program Success Type' section shows 'Success type' as 'Check return code'. The 'Advanced Program Configuration' section shows 'Timeout (minutes)' as 1, 'Wait for completion' checked, 'Program reboots' unchecked, 'Force reboot' unchecked, and 'Environment variables' as an empty list. The 'Program Arguments' section shows 'Response file' as 'DocMgmtSample.prop', 'Log file name' as 'SampleWinPDC.log', and 'Arguments' as '<response file>'. The bottom of the window has a tabbed interface with 'Welcome', 'General', 'Programs', 'Variables', 'Files', 'Libraries', and 'Source' tabs.

User Programs Information

User Programs

Add, remove or edit the user programs defined in the application. The main program is required.

Program Name	Program Type	Program Name
Predeployment Checker	Main Program	

Basic Program Configuration

*Program type: Java program

*Program: com.ibm.iru_samplewin.SampleWinPDC

Custom Program Options

System command: ☐

Java Program Options

Additional classpath:

Advanced Program Configuration

Timeout (minutes): 1

Wait for completion: ☒

Program reboots: ☐

Force reboot: ☐

Environment variables:

Program Success Type

Specify how to determine if the program ran successfully.

Success type: Check return code

Search strings:

Program Arguments

Response file: DocMgmtSample.prop

Log file name: SampleWinPDC.log

Arguments: <response file>

Wellcome | General | **Programs** | Variables | Files | Libraries | Source

Figure 5-6 Programs tab in the Application Wrapper Editor

Variables tab

Figure 5-7 shows the Variables tab. It is used to define/create application variables to help the end-user customize the application's deployment. Using variables, you can collect input from the user in three different formats: string, boolean, and password. You can also constrain the values that users can enter for these parameters by specifying constraints for these parameters. Use the Variables tab on the custom editor to define any variables you need.

Application Variable Information

Application Variables

- String Variable: DocumentDir
- String Variable: DatabaseName
- String Variable: DB2UserId
- Password Variable: password

Basic Variable Configuration

Specify label and help text for the selected variable.

Name: DatabaseName

Label text: %dbNameLabel

Help text: %dbNameHelp

Type: Typical variable

Variable Associations Configuration

Add, remove or edit the associations for the selected variable.

CID File Association: DB2.databaseName

Variable Validation Configuration

Specify validation requirements for the selected variable. You may also specify a default value.

Default value: DOCMGTD7

Required: ☒

Make uppercase: ☐

Make lowercase: ☐

Minimum length: 1

Maximum length: 8

Validation rules:

- Valid prefix: "A"
- Valid prefix: "B"
- Valid prefix: "C"
- Valid prefix: "D"
- Valid prefix: "E"
- Valid prefix: "F"

Figure 5-7 Variables tab in the Application Wrapper Editor

Files tab

As shown in shown in Figure 5-8, the Files tab defines application file information. An application wrapper must list all of the files that need to be transferred to a target computer to run the user programs and application successfully during a deployment. On the Files tab, add all the necessary files to the appropriate file list.

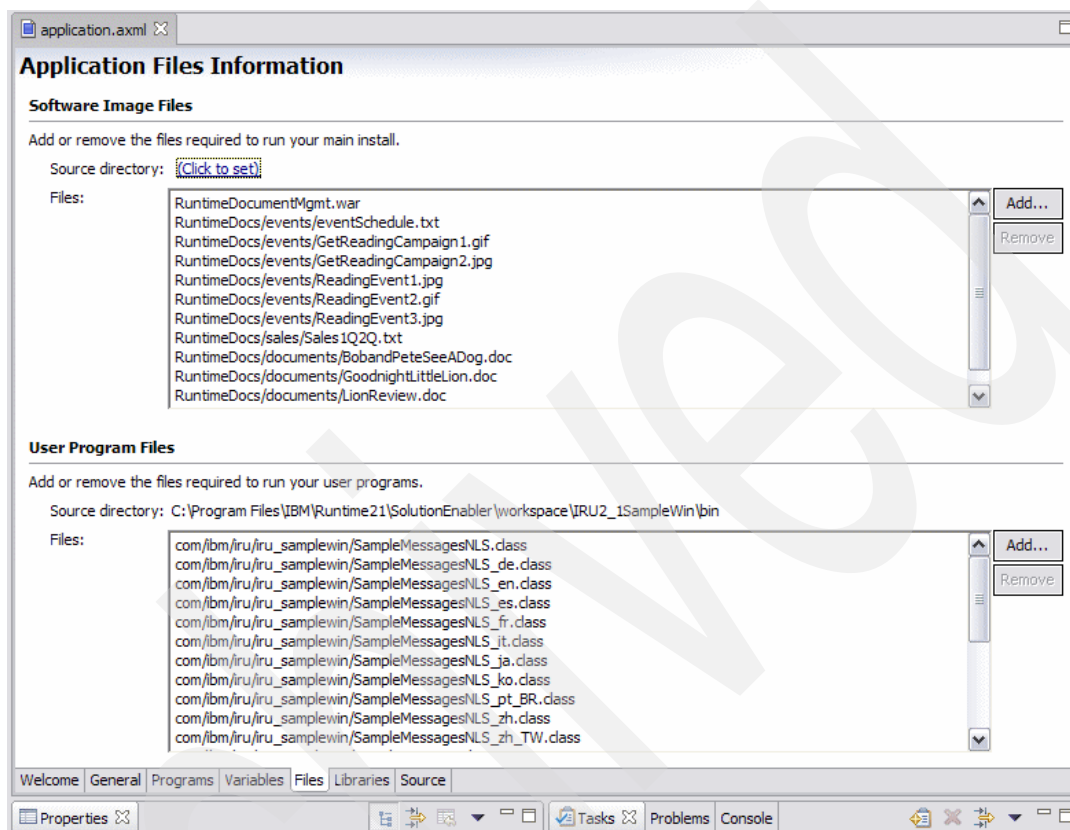


Figure 5-8 Files tab of the Application Wrapper Editor

Software image files are placed in the application's deployment package, which is transferred to the target computer before the entry, main and exit programs run. Software image files can be located anywhere on the development computer. Set the source directory for the software image files and add all the required files to the list.

User program files are placed in the application's user programs package, which is transferred to the target computer before the predeployment checker is run. User program files must be located in the application project within workspace.

Libraries tab

The Libraries tab is shown in Figure 5-9. If user programs require any external libraries (such as JAR files), list them on the Libraries tab.

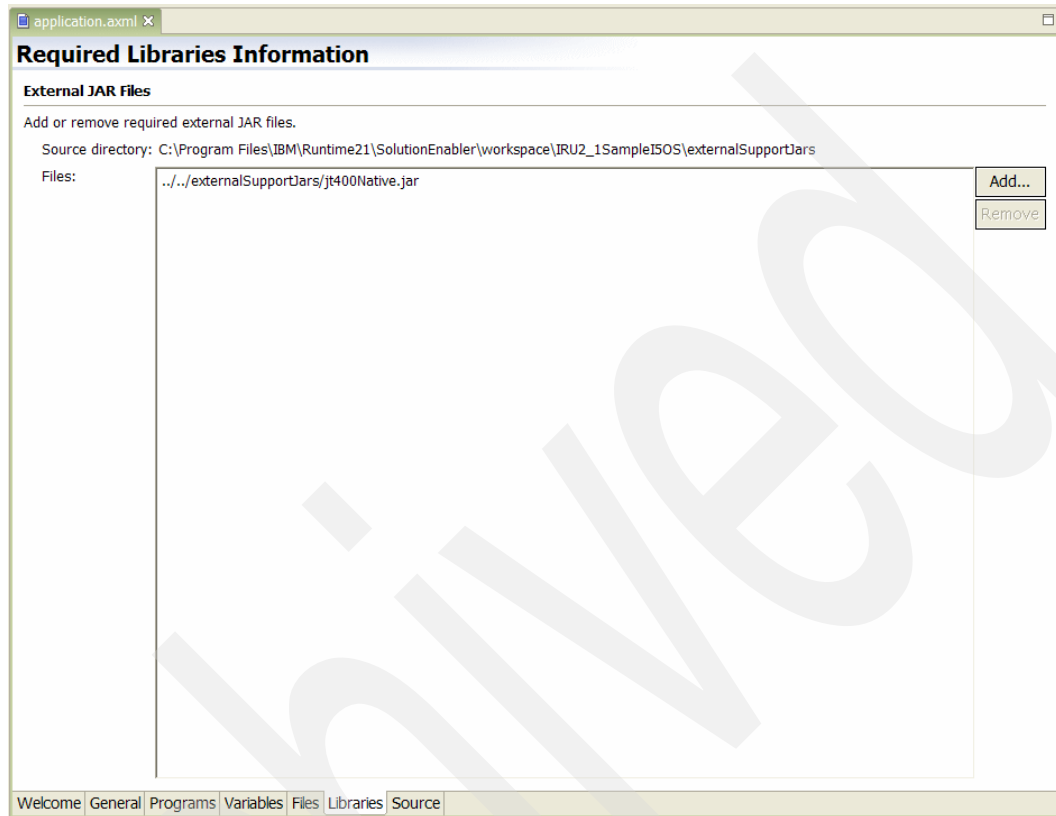


Figure 5-9 Libraries tab of the Application Wrapper Editor

Figure 5-10 shows the Source tab. This gives developers access to the source code of the XML file. However, most of the users should use the GUI interface to set or modify the parameters.

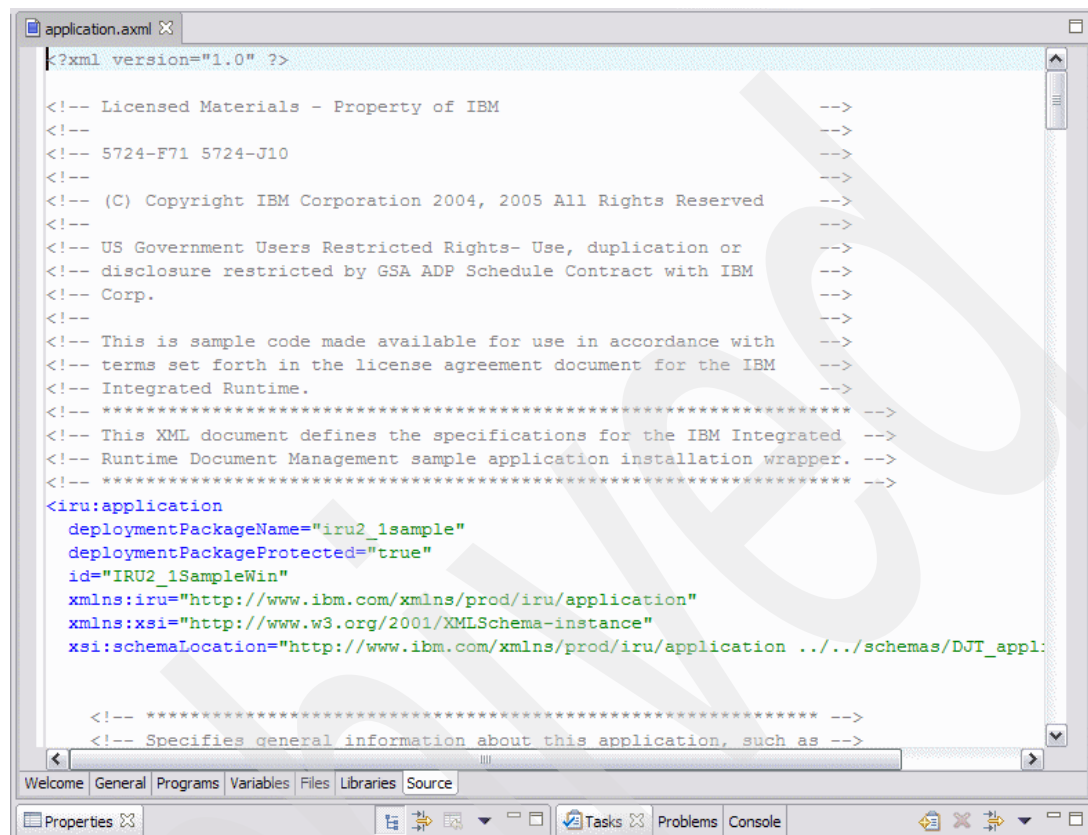


Figure 5-10 Source tab for application.xml

Step: Generate application and deployment packages

Express Runtime Developer prepares and packages the application and associated wrapper in a special package(s). So, in order to test your application wrapper or prepare it for the solution, you need to generate application and deployment packages.

This feature is available from the application project context menu (see Figure 5-11).

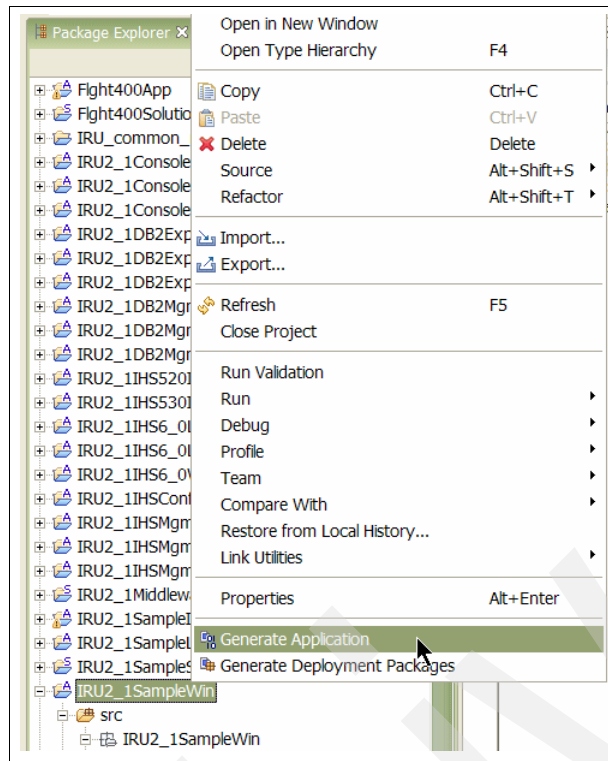


Figure 5-11 Generating application and deployment packages

Step: Add only your application to the solution

For testing of your application wrapper, you need to add your application to a solution. Then you try to deploy this solution to your local system.

If you plan to have a solution with multiple applications, it's easier to test each application separately. In this case, you concentrate on testing and debugging just one application without a complexity of a solution with multiple applications.

In this step you create a dummy solution and add just your application to it. Then you run Deployment Wizard to install your application on the local system. In order to be successful, you need to prepare your local system for such testing. In many cases the application that you test will be part of a solution that includes some other applications.

For example, for a J2EE application, you need to install WebSphere Application Server before you install a J2EE application. For this reason, you need to install all the pre-requisite components onto your local system before you test your application and its dummy solution. Testing of the real solution will come later when you have tested each individual application wrapper and created a solution with all required components. This testing will require a clean system: all the required software components are installed as part of your solution.

5.2.2 Developing a solution wrapper

Now, when you have developed your applications, you need to combine them into a solution. For this reason, you need to create a *solution project*. You create the solution project using the *solution project wizard*.

A solution project has a predefined folder and file structure. A special file called *solution.xml* defines several important parameters of your solution. Express Runtime Developer provides *Solution Wrapper Editor* to view and modify parameters in this file.

The help system in Express Runtime Developer provides a cheat sheet for the solution wrapper development. In order to open this cheat sheet, select **Help** → **Cheat Sheets**. In the pop-up windows, click **Express Runtime** in the left pane and then **Creating a solution wrapper** in the right pane. The help window shown in Figure 5-12 opens.

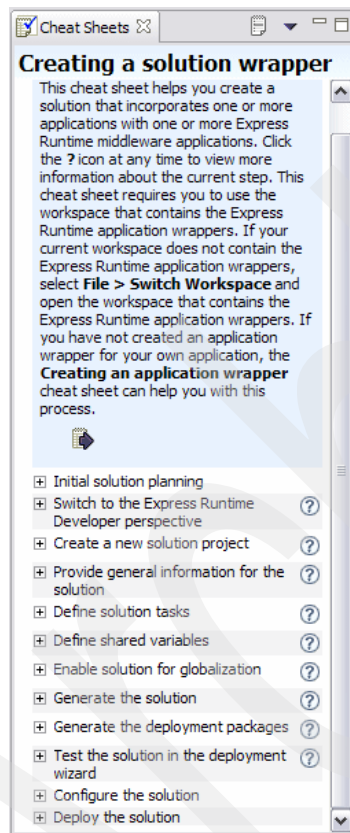


Figure 5-12 Cheat sheet for the solution wrapper development

Creating a solution project

You create a solution project using the solution project wizard, an easy-to-use wizard that requires very few parameters. Most of these parameters are pre-filled with the default values.

When the wizard finishes, you should have a project with the predefined structure; *solution.xml* is part of this structure.

Modifying the solution project

The solution.xml file is edited by using Solution Wrapper Editor. When you open solution.xml in the editor, you should see five tabs along the bottom of the window (see Figure 5-13):

- ▶ Welcome
- ▶ General
- ▶ Tasks
- ▶ Validation
- ▶ Source

Information on the Welcome tab in Figure 5-13 give an overview section with shortcuts to the General, Tasks, Validation, and Source tabs.

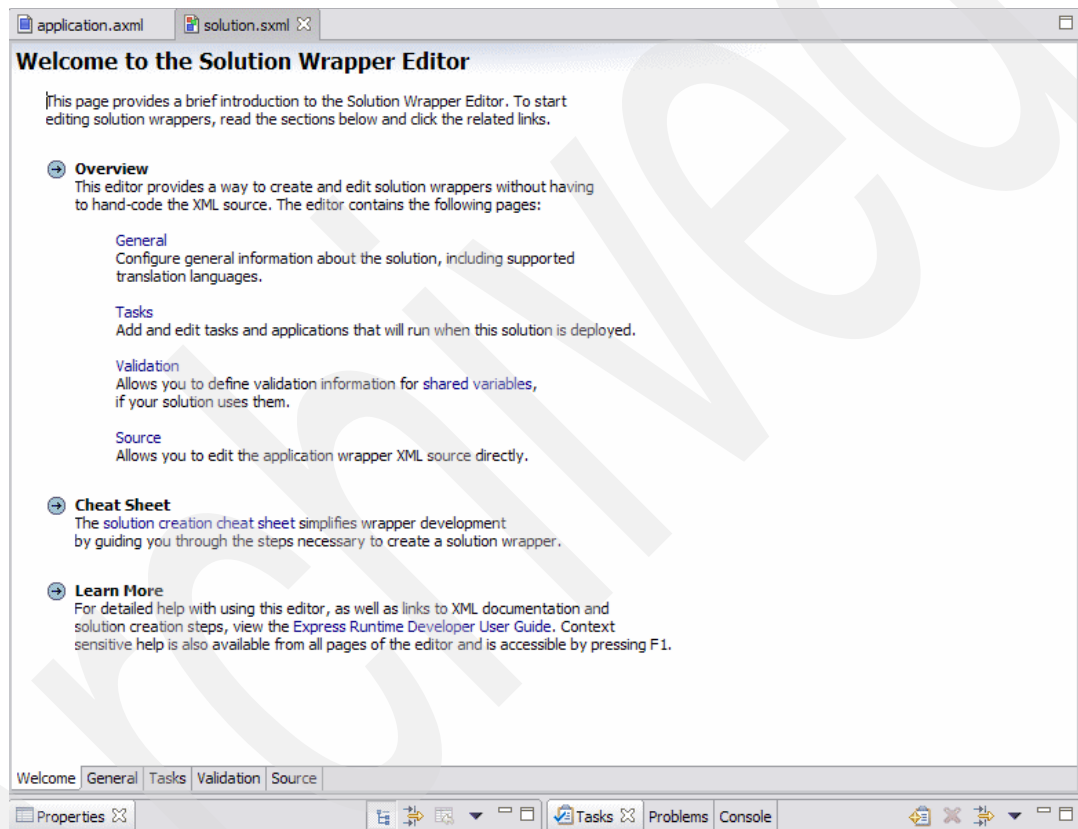


Figure 5-13 Welcome tab for Solution Wrapper in GUI

General tab

Figure 5-14 shows the General tab. The fields that are displayed on the General tab are populated with the information you provided when you created the solution. Now you can provide any additional information for the solution. For most of the fields on the General tab, you can either enter text directly, or provide a translation key that is used to look up translated values in language-specific translation files. To use a translation key, enter a key name and prepend it with a percentage sign (%). For more information see 5.4, “Support for other languages” on page 87.

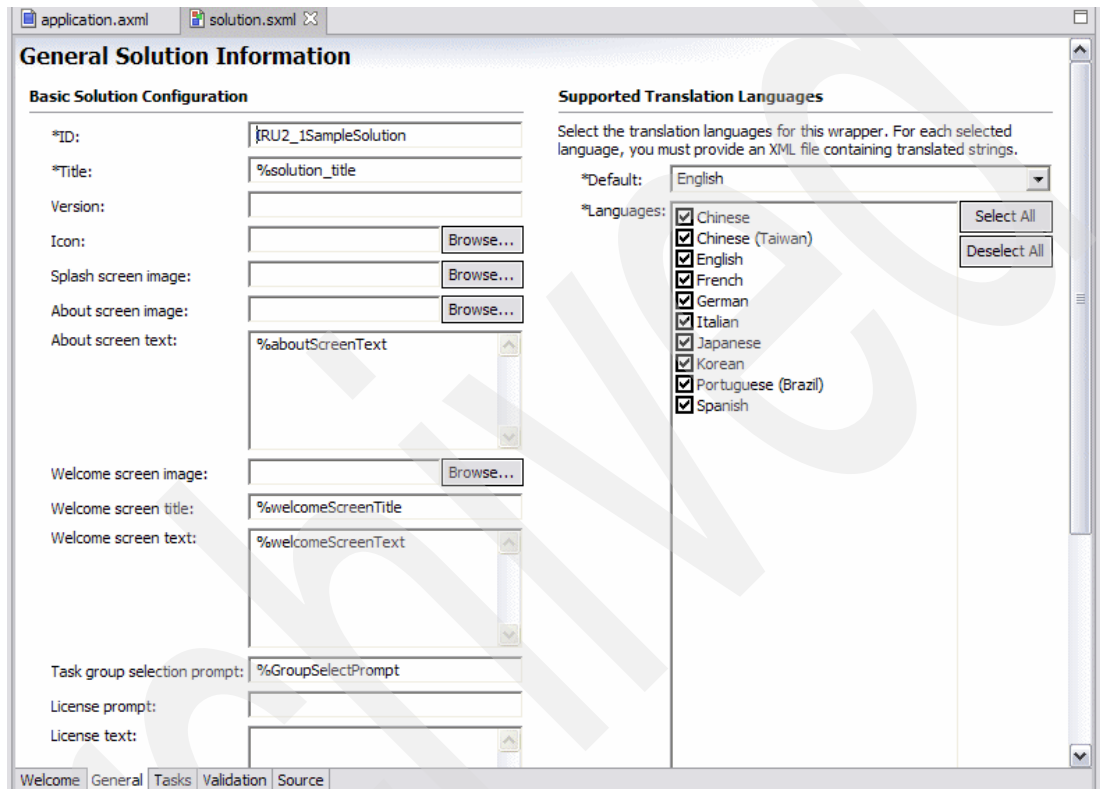


Figure 5-14 General tab for Solution Wrapper in GUI

Tasks tab

Figure 5-15 shows the solutions Tasks tab. All applications must be arranged into the solution tasks. A solution consists of one or more tasks. There are two types of tasks:

- Manual task
- Install task

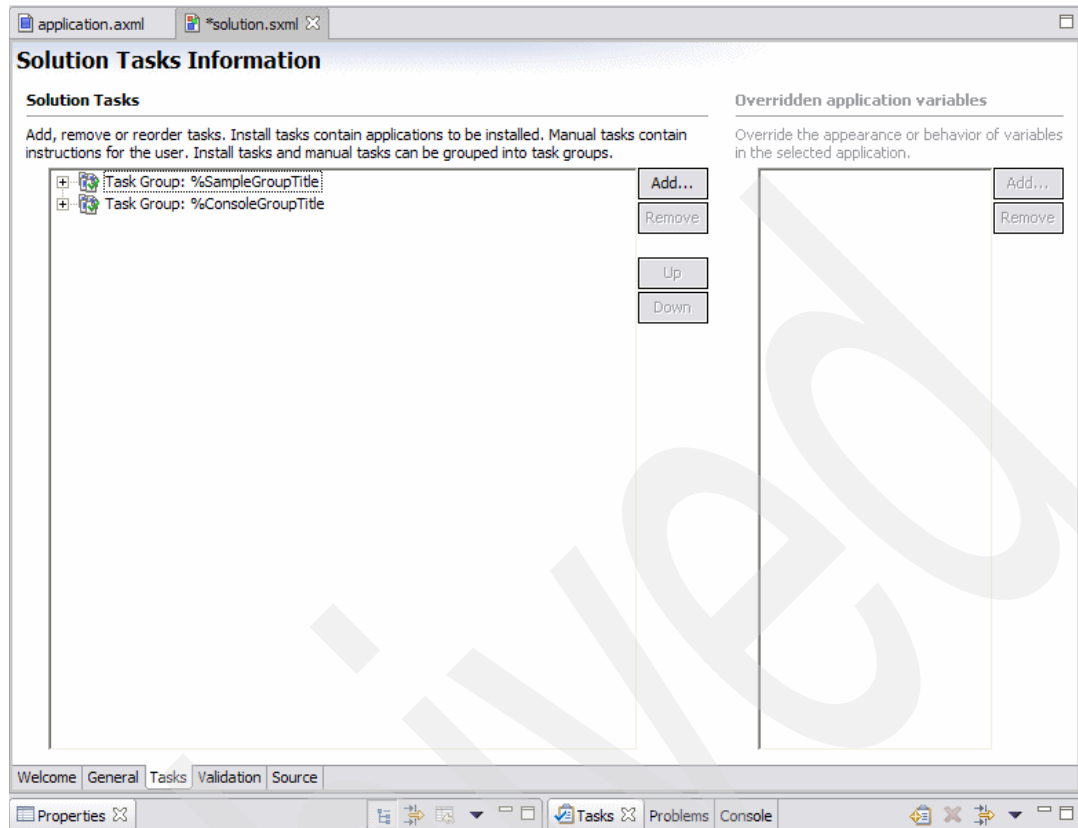


Figure 5-15 Tasks tab for Solution Wrapper in GUI

You add a task by clicking the **Add** button. The wizard takes you through the different set of steps, based on the type of the selected solution task.

Another set of information that you define on this tab is *Overridden application variables*.

The following sections describe the Tasks tab in more detail.

Manual task

Manual tasks display instructions that you can customize during deployment of the solution, and wait for the end user to indicate that the instructions have been followed. For example, a manual task might instruct the end user to copy a script generated on one system during installation to another system and run that script.

Install task

Install tasks consist of one or more applications to be deployed to one or more target computers. To add any application to the install task, the application has to be part of the Express Runtime Developer workspace. You invoke the *Add Application wizard* to add one or more applications to the install task.

An install task is bound to a specific deployment platform, which means that all applications in the install task have to be for the same platform. To illustrate this rule, consider a case where you need to develop a solution for the Linux and i5/OS platforms. In this scenario, you need to create at least two tasks: one task that includes the applications for Linux and another task that includes the applications for i5/OS. At the deployment time, a user may select which install task to apply to the target system.

Task group

For the convenience of the deployment process, you may organize your tasks in two or more task groups. The usage of task groups is optional.

The decision for using a task group should be made by a solution developer based on the deployment pattern of the solution. As an example, look at Figure 5-15. This is the IRU2_1SampleSolution sample solution. It defines two task groups: one for deploying the middleware and another for deploying the console for Express Runtime. When you start deploying this solution, you should see a window similar to one shown in Figure 5-16.

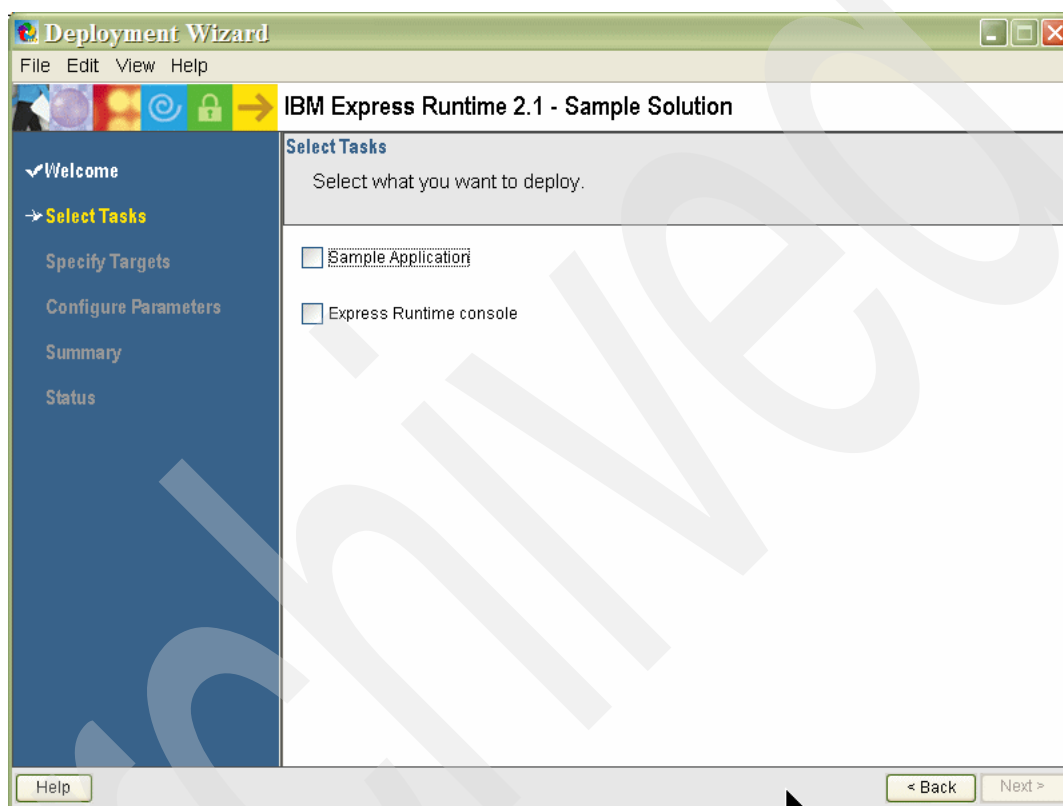


Figure 5-16 The deployment wizard window

If you select the **Sample Application** checkbox and click **Next**, you will see four install tasks (see Figure 5-17).

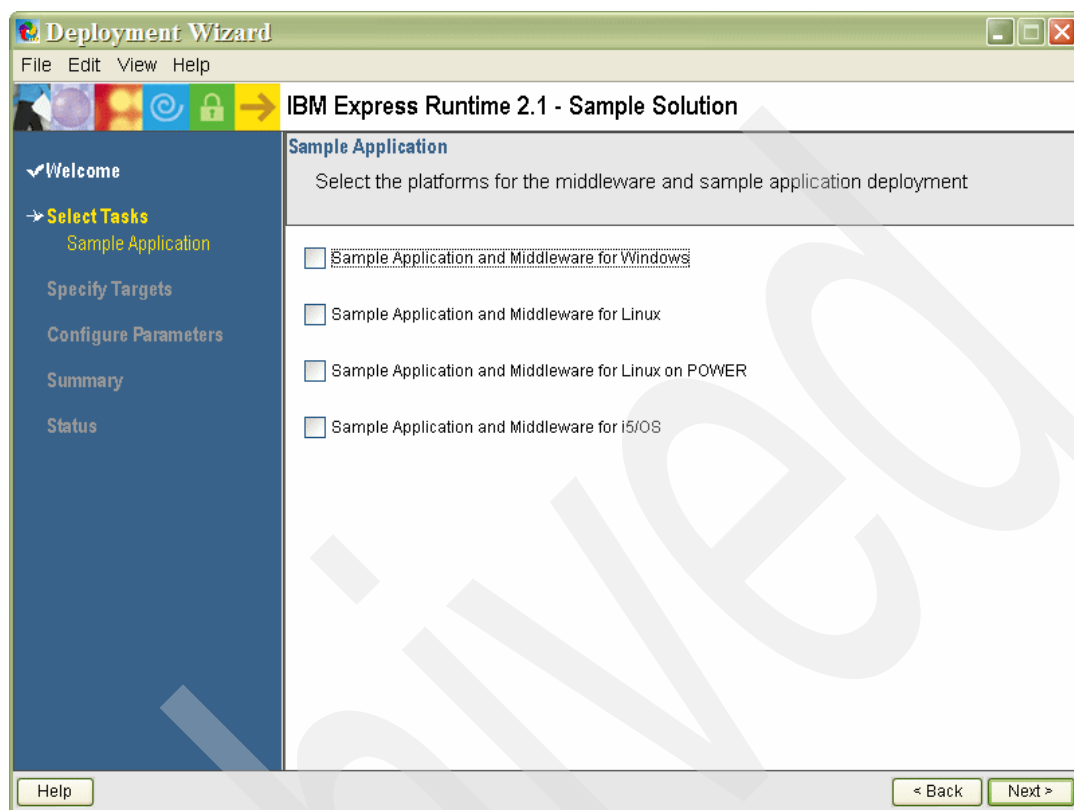


Figure 5-17 The install tasks of the sample application

The same four install tasks are shown in the solution.xml file (see Figure 5-18). However, the translation keys are replaced during deployment time (for more information, see 5.4, “Support for other languages” on page 87).

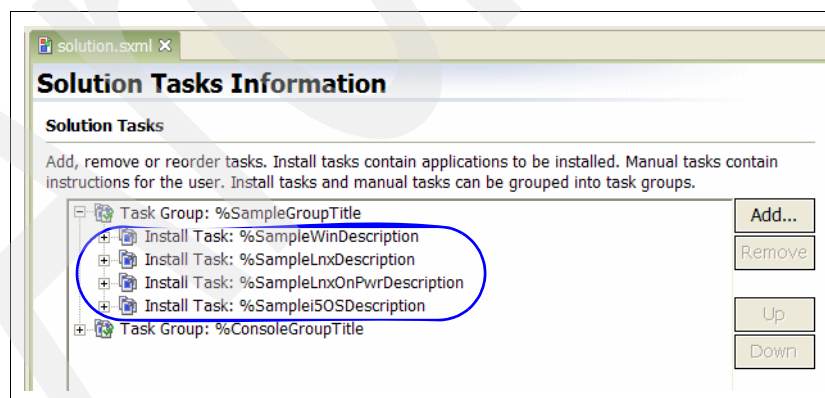


Figure 5-18 Install tasks

Finally, if you expand the first install task, you should see several applications for the Windows platform (see Figure 5-19).

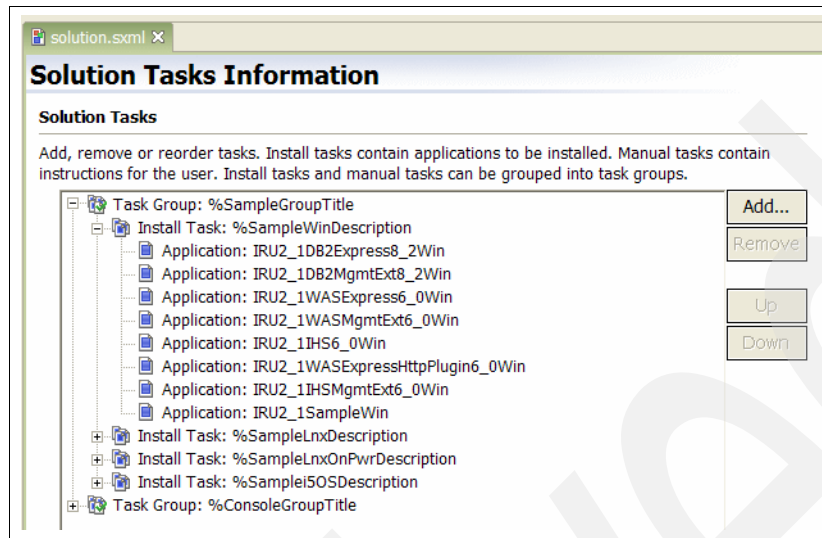


Figure 5-19 Applications in the install task

This example is a good illustration of how you can organize the task groups and install tasks for your solution.

Overridden application variables

When you develop an application, you may create several variables (see “Variables tab” on page 72). All of the variables defined within an application are displayed as editable fields on the configuration parameters panel in Deployment Wizard. When an application is added to a solution, variables within the application can be hidden or made read-only for a particular configuration parameters panel. This adds the flexibility to override the default variables behavior based on a specific solution requirement.

After you add an application to the solution tasks list, you can select this application and click the **Add** button next to the *Overridden application variables* box. The Override Application Variables wizard starts. It shows only the variable defined within the selected application. You need to add only the variable whose behavior and value you want to override.

As an example, look at Figure 5-20. It shows that for the IRU2_1WASMgrExt6_0Win application in the sample solution, we have overridden five application variables.

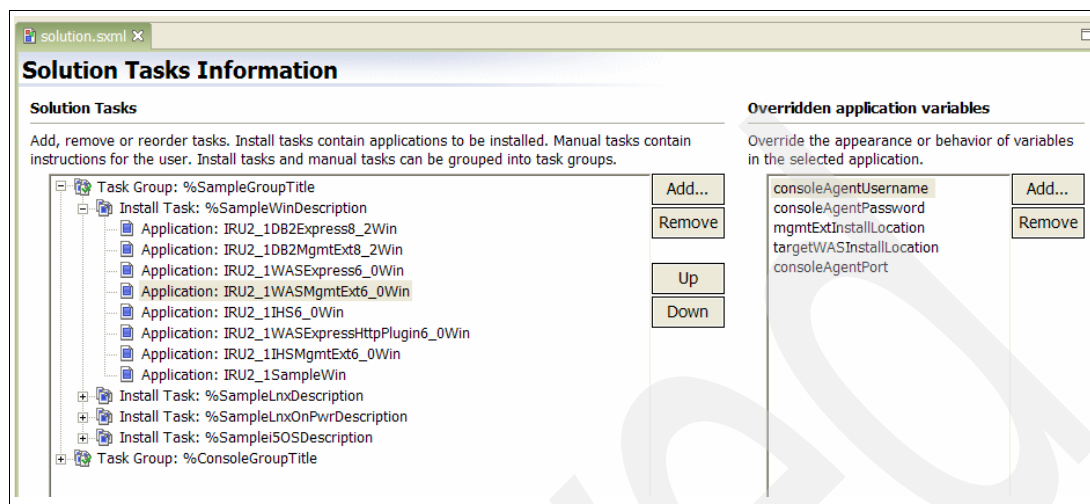


Figure 5-20 Overridden application variables

For each of the overridden variables, you specify two parameters:

- ▶ Appearance
- ▶ Behavior

For the appearance, you may select from the following options:

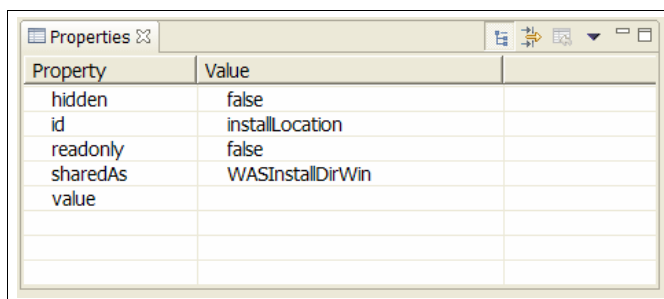
- ▶ Editable. This is the default option. A user can change the value during the deployment time.
- ▶ Hidden. Not exposed to a user.
- ▶ Read-only. A user can't change its value.

For the behavior, you may select from the following options:

- ▶ Application-defined. This is the default option.
- ▶ Modify the default value of <variable name>. You may decide to change or define the default value for this variable.
- ▶ Share the value of <variable name> with other variables. Sometimes, applications need to access the variable values of another application in the same solution. For example, when you deploy a solution that installs a J2EE application and configures a database for this application, you use the database name variable in both applications.

You may share the database name variable in your DB2 application with the J2EE application. In this case, a value of this variable can be accessed by both applications. As a result, the user does not need to enter the database name multiple times in Deployment Wizard. This also provides another benefit by reducing the chance of mistyping the database name. More details are given in "Validation tab" on page 85.

If you need to modify any of the options, select the variable in the Overridden application variables list. All the parameters for this variable are shown in the Properties view, which is located at the bottom of the Express Runtime Developer window (see Figure 5-21).



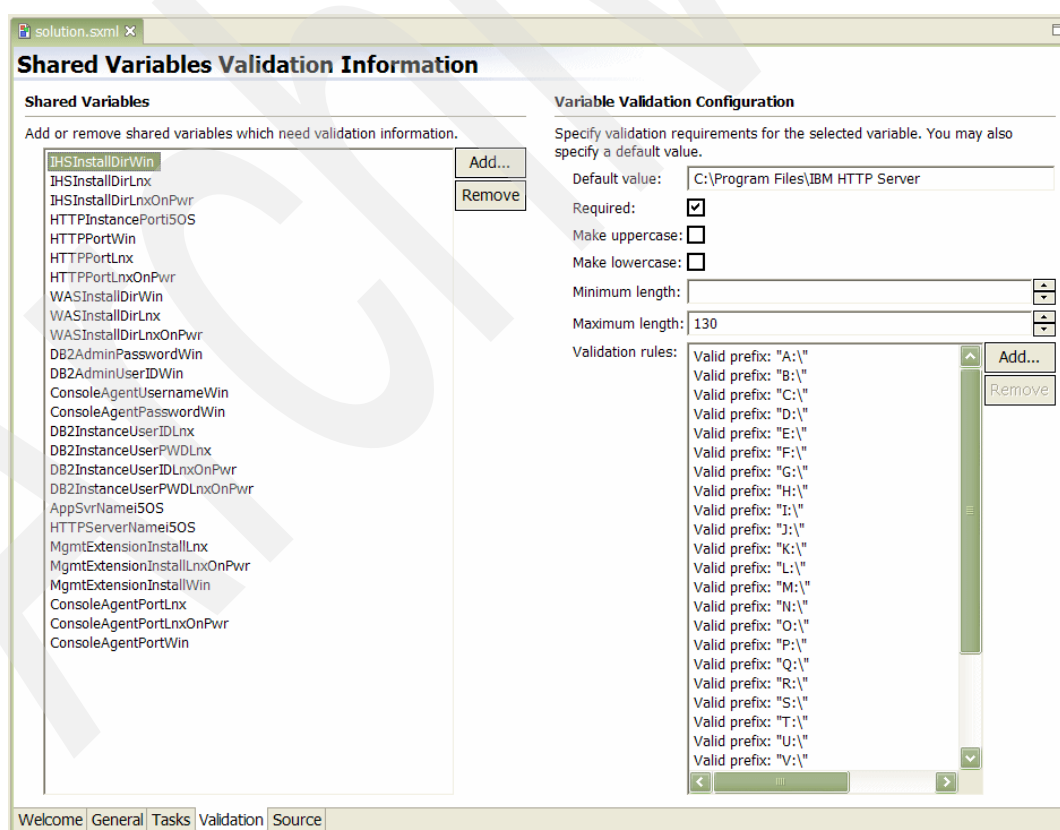
Property	Value
hidden	false
id	installLocation
readonly	false
sharedAs	WASInstallDirWin
value	

Figure 5-21 The Properties view

With this flexibility, you can modify the application variables appearance and behavior based on the needs of your solution.

Validation tab

Figure 5-22 shows the Validation tab. On this tab you can override the validation rules for the shared variables (see “Overridden application variables” on page 83). The rules specified on this tab override the rules defined in the application wrapper (see “Variables tab” on page 72 for more information).



Shared Variables Validation Information

Shared Variables

Add or remove shared variables which need validation information.

IHSInstallDirWin
IHSInstallDirLnX
IHSInstallDirLnXOnPwr
HTTPInstancePort50S
HTTPPortWin
HTTPPortLnX
HTTPPortLnXOnPwr
WASInstallDirWin
WASInstallDirLnX
WASInstallDirLnXOnPwr
DB2AdminPasswordWin
DB2AdminUserIDWin
ConsoleAgentUsernameWin
ConsoleAgentPasswordWin
DB2InstanceUserIDLnX
DB2InstanceUserPWLnX
DB2InstanceUserIDLnXOnPwr
DB2InstanceUserPWLnXOnPwr
AppSvrName50S
HTTPServerName50S
MgmtExtensionInstallLnX
MgmtExtensionInstallLnXOnPwr
MgmtExtensionInstallWin
ConsoleAgentPortLnX
ConsoleAgentPortLnXOnPwr
ConsoleAgentPortWin

Add...
Remove

Variable Validation Configuration

Specify validation requirements for the selected variable. You may also specify a default value.

Default value: C:\Program Files\IBM HTTP Server

Required: ☒

Make uppercase: ☐

Make lowercase: ☐

Minimum length:

Maximum length: 130

Validation rules:

Valid prefix: "A:\"
Valid prefix: "B:\"
Valid prefix: "C:\"
Valid prefix: "D:\"
Valid prefix: "E:\"
Valid prefix: "F:\"
Valid prefix: "G:\"
Valid prefix: "H:\"
Valid prefix: "I:\"
Valid prefix: "J:\"
Valid prefix: "K:\"
Valid prefix: "L:\"
Valid prefix: "M:\"
Valid prefix: "N:\"
Valid prefix: "O:\"
Valid prefix: "P:\"
Valid prefix: "Q:\"
Valid prefix: "R:\"
Valid prefix: "S:\"
Valid prefix: "T:\"
Valid prefix: "U:\"
Valid prefix: "V:\"

Add...
Remove

Figure 5-22 Validation tab for Solution Wrapper

Source tab

Figure 5-23 shows the source tab. This gives developers access to the source code of the solution.xml file. However, in most cases, you change the parameters in this file by using other tabs.

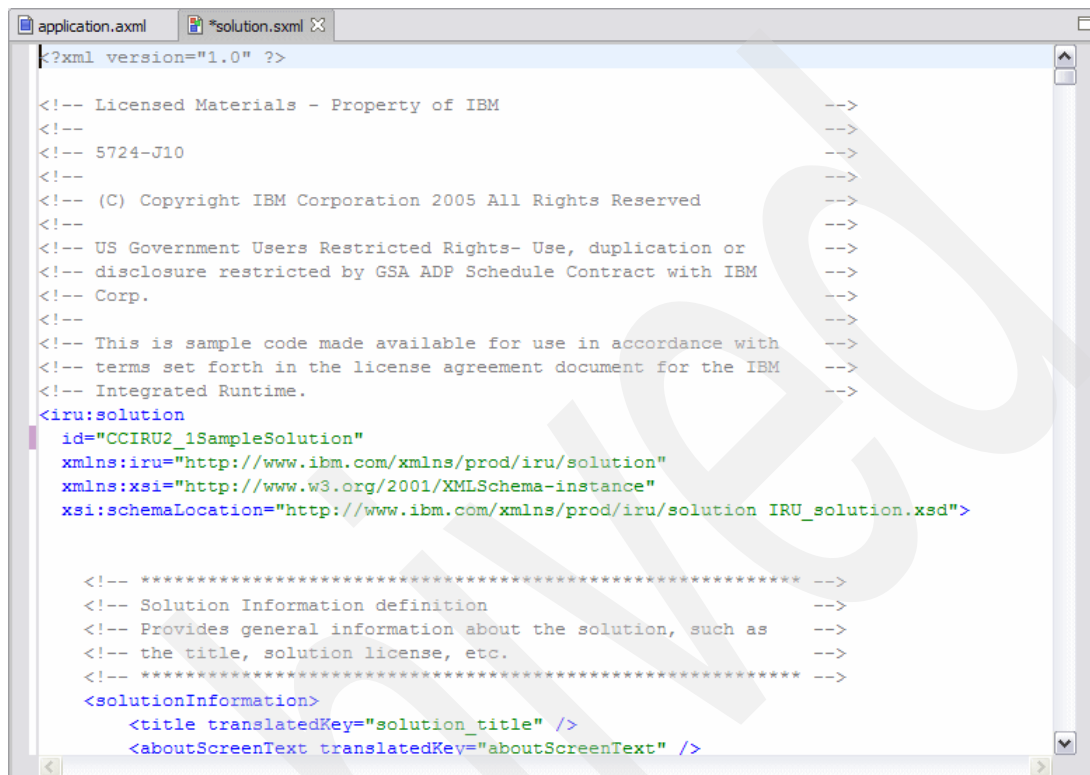


Figure 5-23 Source tab for solution.xml

5.3 Support Framework API

Java programs are typically used for writing the user programs. Express Runtime Developer provides the Support Framework to aid in developing user programs for the supported platforms.

The Support Framework package is located at:

```
<ER Install dir>\Runtime21\SolutionEnabler\Support_Framework
```

You can display the Javadoc by pointing your browser to:

```
<ER Install dir>\Runtime21\SolutionEnabler\Support_Framework\index.html
```

The best way to master the support API is by examining the sample applications provided in the Express Runtime Developer workspace.

5.4 Support for other languages

Express Runtime V2.1 allows you to use one of the following languages for Deployment Wizard:

- ▶ English
- ▶ Spanish
- ▶ French
- ▶ German
- ▶ Italian
- ▶ Brazilian Portuguese
- ▶ Japanese
- ▶ Simplified Chinese
- ▶ Traditional Chinese
- ▶ Korean

You may select the supported languages when you go through the application project wizard. You can also change this selection at a later time by modifying the application.xml file.

The language specific information is stored in the language XML files. For each supported language you need to have a separate file. Figure 5-24 shows the XML files for each supported language.

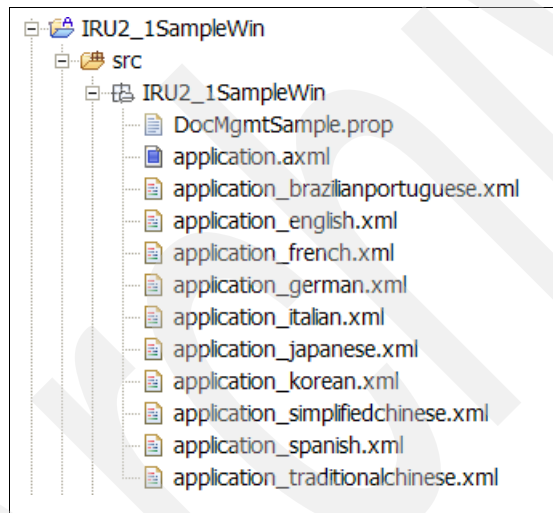


Figure 5-24 Language specific files

Each of these files has the same set of tags. Each tag represents a translated version of a string. For example, the value of the tag <dbNameLabel> in application_english.xml is:

Database Name

The value of this tag in application_german.xml is:

Datenbankname

As the result, Deployment Wizard uses a file that is specific to the selected language, and at runtime, replaces the translation key with its value from the language-specific file.

Important: This support is for the deployment process only. If you need to support multiple languages in your J2EE application, for example, you should do this during the application development process.

To illustrate this information, look at Example 5-1. It shows the English version of the file.

Example 5-1 application_english.xml

```
<IRU2_1SampleWin>
  <configureText>Provide information for the configuration parameters and click
  Next.</configureText>
  <providerName>IBM</providerName>
  <name>Express Runtime Publishing Document Manager</name>
  <prompt>Provide the fully-qualified path name where the Express Runtime Publishing
  Document Manager installation image is located (for example, C:\download\IBM).</prompt>
  <DocDirLabel>Document Directory</DocDirLabel>
  <DocDirHelp>The directory where you want to store the application documents.</DocDirHelp>
  <dbNameLabel>Database Name</dbNameLabel>
  <dbNameHelp>The name of the database to be created and used by this application. If the
  database already exists, it is dropped.</dbNameHelp>
  <DB2UserIdLabel>DB2 Administrator UserID</DB2UserIdLabel>
  <DB2UserIdHelp>The Administrator id used to connect to DB2.</DB2UserIdHelp>
  <DB2PasswordLabel>DB2 Administrator Password</DB2PasswordLabel>
  <DB2PasswordHelp>The Password used with the id specified to connect to
  DB2.</DB2PasswordHelp>
</IRU2_1SampleWin>
```

Example 5-2 shows the same file, but in German translation.

Example 5-2 application_german.xml

```
<IRU2_1SampleWin>
  <configureText>Geben Sie Informationen für die Konfigurationsparameter an, und klicken
  Sie auf 'Weiter'. </configureText>
  <providerName>IBM</providerName>
  <name>Express Runtime Publishing Document Manager</name>
  <prompt>Geben Sie den vollständig qualifizierten Pfadnamen an, unter dem sich das
  Installationsimage von Express Runtime Publishing Document Manager befindet (z. B.
  C:\download\IBM). </prompt>
  <DocDirLabel>Dokumentverzeichnis</DocDirLabel>
  <DocDirHelp>Das Verzeichnis, in dem die Anwendungsdokumente gespeichert werden sollen.
  </DocDirHelp>
  <dbNameLabel>Datenbankname</dbNameLabel>
  <dbNameHelp>Der Name der Datenbank, die von dieser Anwendung erstellt und verwendet
  werden soll. Dieses Feld erscheint nicht, wenn die Datenbank bereits existiert.
  </dbNameHelp>
  <DB2UserIdLabel>Benutzer-ID des DB2-Administrators</DB2UserIdLabel>
  <DB2UserIdHelp>Die Administrator-ID, die zur Anmeldung an der DB2-Instanz verwendet
  wird.</DB2UserIdHelp>
  <DB2PasswordLabel>Kennwort des DB2-Administrators</DB2PasswordLabel>
  <DB2PasswordHelp>Das Kennwort, das mit der angegebenen ID verwendet wird, um die
  Verbindung zu DB2 herzustellen.</DB2PasswordHelp>
</IRU2_1SampleWin>
```

A solution project applies the same technique to provide a support for different languages (see Figure 5-25).

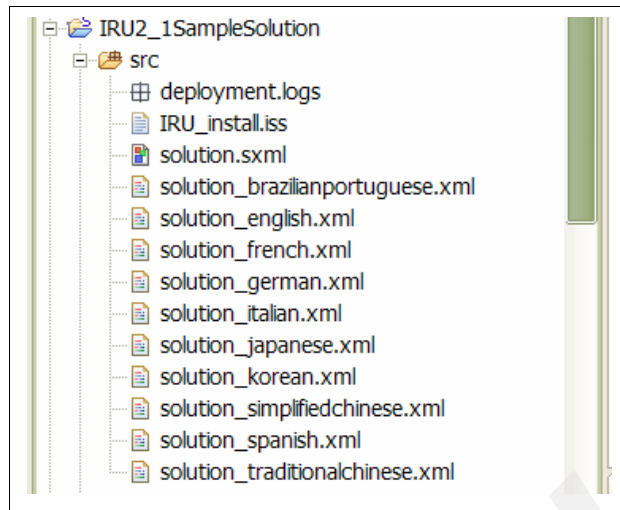


Figure 5-25 Language specific files in the solution project

In addition to supporting multiple languages during the deployment, you can also display information about your solution, license, and readme in all the supported languages. You need to add the language specific files to the corresponding folder under the solution project. See the folder structure for the license files in Figure 5-26.

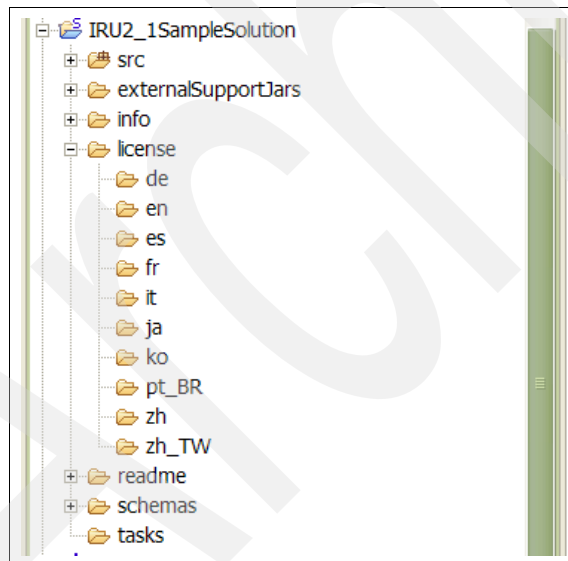


Figure 5-26 License information

Archived

Developing a wrapper

Now that you have a firm understanding of what a wrapper is and how it is structured, we take you to the next level of developing wrappers for your application.

IBM Express Runtime provides the capability for ease of deploying ISV solutions. In this chapter, we show you the process for creating the needed wrappers to be able to deploy your application and the necessary middleware successfully. We work on several real-life applications to demonstrate this.

Note: For this chapter, *ER install dir* and *<ER workspace dir>* are referenced frequently to reduce the length of the total path. For Windows, these variables correspond to the following paths:

- ▶ *ER install dir* is your IBM Express Runtime V2.1 installation directory
C:\Program Files\IBM\Runtime21
- ▶ *<ER workspace dir>* is your IBM Express Runtime V2.1 workspace folder
C:\Program Files\IBM\Runtime21\SolutionEnabler\workspace

6.1 Overview of the sample applications

To illustrate the step-by-step procedure of creating wrappers, we are using three applications targeted to three different platforms. This section provides a high level description of the three applications:

- ▶ Trade6 on Windows platform
- ▶ Trade6 on Power Linux
- ▶ Flight400 WebFacing application on i5/OS

Disclaimer: All values in these examples are for demonstration purpose only. Apply appropriate values for your own applications.

6.1.1 Trade6

Trade6 is an IBM created J2EE application modeled after an online brokerage. This application leverages many aspects of J2EE V1.4 and of WebSphere Application Server V6, such as servlets, JSPs, EJBs, JDBC, and Web Services, to provide a set of user services such as login/logout, stock quotes, buy, sell, account details, and so forth.

For more information on Trade6, you may refer to Chapter 2 of the redbook *Application Development using the Versata Logic Suite for WebSphere*, SG24-6510.

In order to install Trade6 as part of a solution in Express Runtime V2.1, the following files should be available (see Appendix D, “Additional material” on page 435 for more information):

- ▶ Trade.ear: The Trade6 application in EAR format.
- ▶ Table.ddl: The Data Definition Language file for creating the tables used in Trade6.
- ▶ DB2Script.bat: Batch file to create the Trade6 Database and execute Table.ddl.
- ▶ CheckAppInstall.jacl: The script checks if an application exists in the list of installed applications in a selected WebSphere Application Server (WAS) profile. This is used by the user programs to verify if Trade6 exists in the target system prior to deployment.
- ▶ SetupProcs.jacl: This script file is used to set the values specified in the application response file for deployment and other jacl script requirements.
- ▶ WebSphereScript.jacl and WebSphereConfigProcs.jacl: The script files to configure the environment needed by Trade6 in WebSphere (for instance: JAAS authorization, DB2 JDBC Provider, DB2 Data Source, JMS Resources, and so on).

We will be creating two wrappers for Windows and Power Linux for this J2EE application.

6.1.2 WebFacing application

The Flight400 application has been created using the IBM WebFacing Tool. The IBM WebFacing Tool is part of the IBM WebSphere Development Studio Client for iSeries. It provides a quick way to Web-enable existing iSeries applications with minimal, if any, modifications to the original host application. The IBM WebFacing Tool provides a simple mechanism to re-face the existing 5250 applications with Hypertext Markup Language (HTML) user interfaces. The result of this re-facing process is a J2EE compliant application packaged in the EAR file.

To get more information about this WebFacing application, see Chapters 4 and 5 in the redbook, *Mastering the IBM WebFacing Tool*, SG24-6331.

6.2 Developing the Trade6 wrapper for Windows

To package Trade6 in a solution, we need to create the necessary wrappers to define installation and configuration steps. We use Express Runtime V2.1's wizard and GUI features for this purpose.

The wrapper for Trade6 with contains 2 user programs: predeployment checker (PDC) and Main. The Trade6 application comes with the wsadmin scripts that you run in the interactive mode. They are not suitable for utilizing them as part of the wrapper.

So, we had to create them outside of the Express Runtime Developer environment. Our next step was to test them. After successful test we are ready to start working with Express Runtime Developer.

Note: If you want to view this sample wrapper, download and import its files into Express Runtime Developer. Refer to Appendix D, "Additional material" on page 435, for the download instructions.

We provide two ZIP files for the Trade6 solution for Windows.

- ▶ *Trade6WinSolution.zip* contains the solution for a single Windows system. You'll find the description of the development process for this solution in this section.
- ▶ *Trade6WinSolutionMultisystem.zip* contains the Trade6 application and middleware components, but is designed to be deployed to multiple systems. This solution also includes the Console for Express Runtime.

6.2.1 Creating the Trade6 application project

To develop a wrapper, you need to create an application project. The following steps describe how to create the Trade6 application project:

1. Start Express Runtime Developer by selecting **Start → Programs → IBM Express Runtime 2.1 → Express Runtime Developer**. You should be in the Express Runtime Developer perspective (see Figure 6-1).

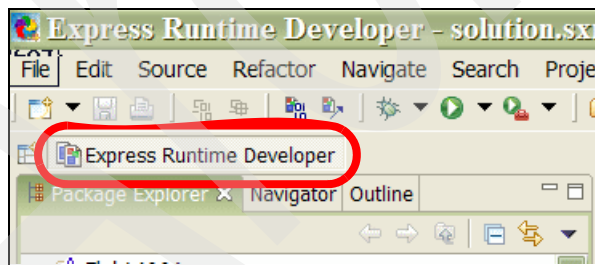


Figure 6-1 Express Runtime Developer perspective

2. Create a new Express Runtime application project.
 - a. In the Express Runtime Developer window, select **File** → **New** → **Application Project** (see Figure 6-2).

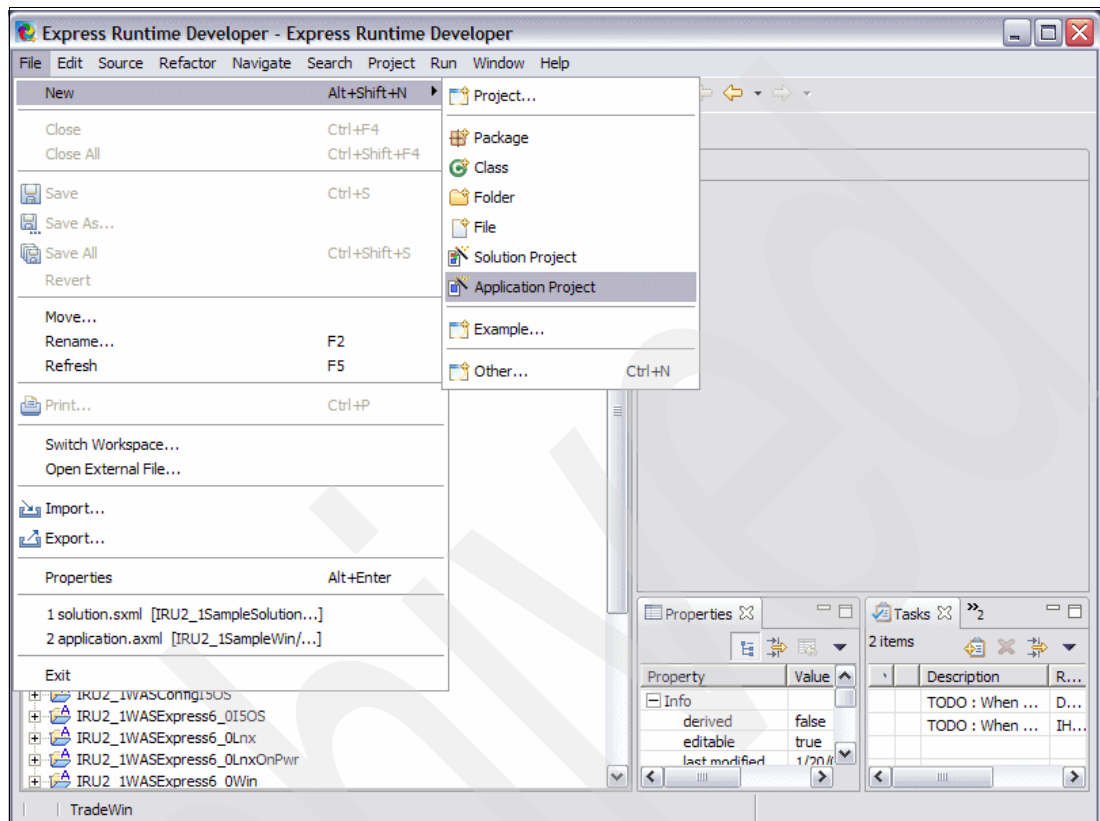


Figure 6-2 Creating a new application project

- b. Type TradeWin as the Project Name and click **Next** as indicated in Figure 6-3.

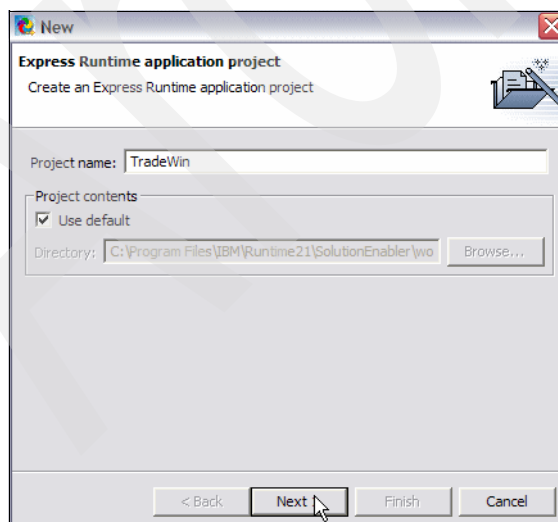


Figure 6-3 New project name

- c. In the next window, use the following values and click **Finish** as shown in Figure 6-4:
- Application ID: TradeWin
 - Version: 1.0
 - Installation time: 20. This is the approximate installation time in minutes. This value is used by Deployment Wizard to show the installation progress.
 - Operating System: Windows
 - Default Language: English
 - Wrapper file name: application.xml. This is the default name. You may come up with your own name. But we recommend to leave it as is.

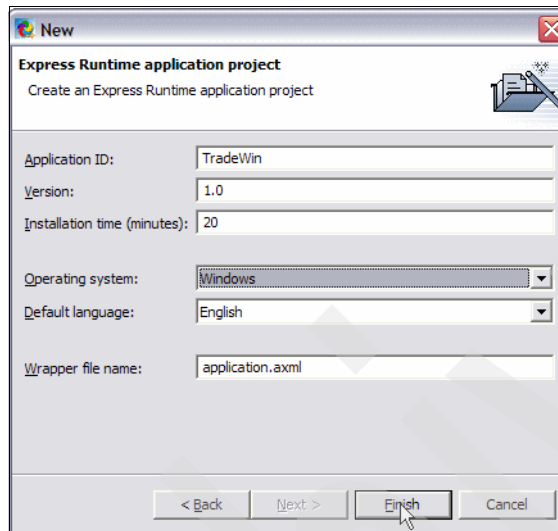


Figure 6-4 New application id and basic information

Note: In this example, we are deploying Trade6 on Windows. If your application supports a different platform, you can select a different operating system from the Operating system field (see Figure 6-5).

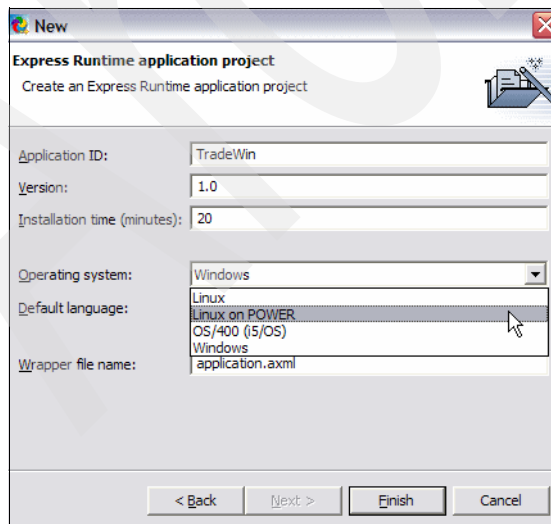


Figure 6-5 Selecting supported operating system

3. The Express Runtime Developer window now displays the Application Wrapper Editor Welcome page in the right pane. The Package Explorer panel shows the newly created application folder named TradeWin (Figure 6-6).

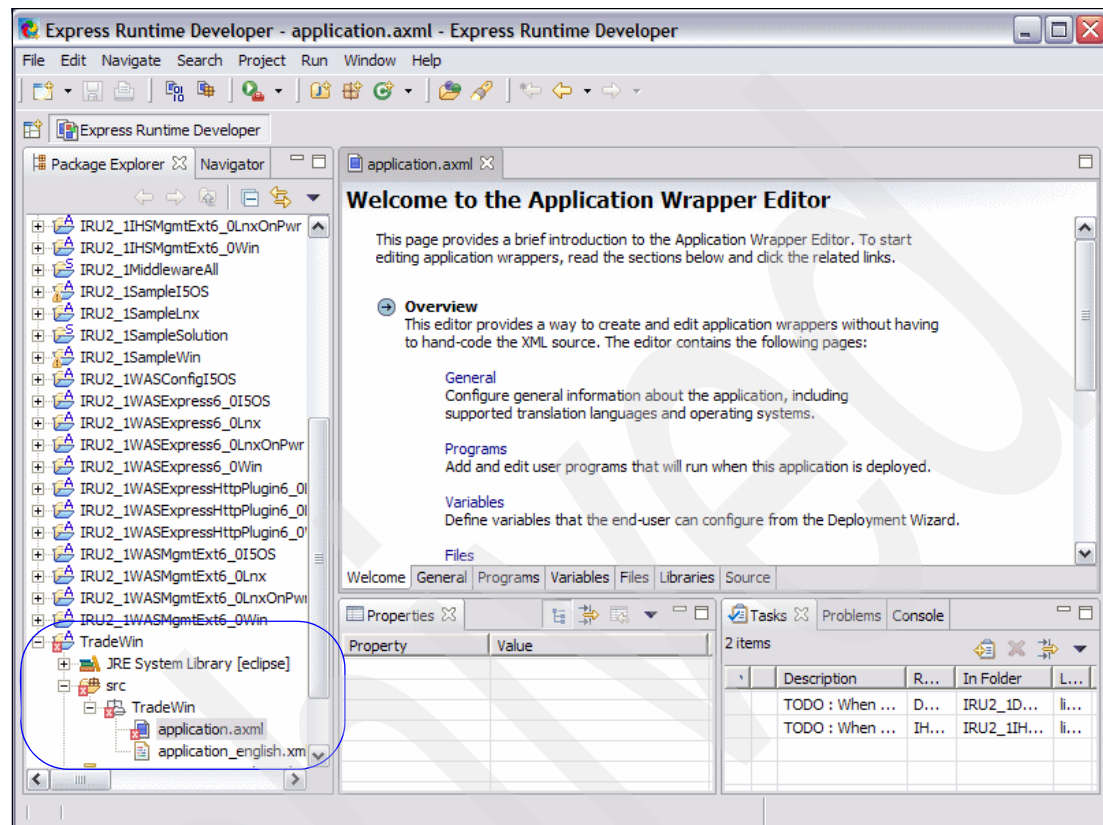


Figure 6-6 Application Wrapper Editor Welcome page

- a. Click the **General** tab. The page in Figure 6-7 appears. This gives you the fields where you specify the general information for your application project. Set the following values as shown:

- Application Name: TradeWin
- Installation time: 1.0
- Provider Name: IBM
- Default Language: English
- Supported Operating Systems: Windows

Note: Remember that you can use the translation keys as values for the parameters. See “General tab” on page 70.

The screenshot shows the 'General Application Information' dialog box. The 'Basic Application Configuration' section contains the following fields: *ID (TradeWin), *Application name (TradeWin), *Version (1.0), Installation time (minutes) (20), Provider name (IBM), and License text. The 'Advanced Application Configuration' section contains: Deployment package protected (unchecked), Deployment package name, and Configuration instructions. The 'Supported Translation Languages' section on the right has a 'Default' dropdown set to 'English' and a list of languages with checkboxes: Chinese, Chinese (Taiwan), English (checked), French, German, Italian, Japanese, Korean, and Portuguese (Brazil). The 'Supported Operating Systems' section has a list of OSes with checkboxes: Linux, Linux on POWER, OS/400 (i5/OS), and Windows (checked). The bottom of the dialog shows a tabbed interface with 'General' selected.

Figure 6-7 General application information

Note: Should your application support other languages and operating systems, you can use this page to select the appropriate options.

- b. Click the **Programs** tab. This brings up the User Programs Information page.

Note: The Main Program is required in an application project. However, there is no program associated with it by default.

In our Trade6 application project, we use two user programs: a Pre-deployment Checker, and the Main Installation program.

The Pre-deployment Checker examines the target system for any prerequisite on the target system.

The Main program performs the tasks to deploy the Trade6 application. Such tasks are creating a DB2 table, and invoking wsadmin scripts to configure WAS (for example: configure resources, install EAR, etc.).

- i. In the same page under the User Programs section, click the program type **Main Program**.
- ii. Leave the value Java Program in the program type parameter under Basic Program Configuration section.
- iii. In the Program field, type `com.trade.TradeWinMain`. This step associates the TradeWinMain class as the Main Program.

Note: The program TradeWinMain does not exist yet. We will create this when we get to section “Creating Trade6 user programs” on page 113. If you choose to create the program (class) first, then you can browse to it instead of typing the name.

- iv. On the Program Arguments section, type in `Trade.prop` in the Response file field.

Note: The Trade.prop file does not exist at this time. We will create this in section 6.2.3, “Creating Trade6 user programs” on page 113.

- v. Type in `TradeWinMain.log` for the Log file name.
- vi. In the Arguments field, click **Add** in the Add Argument window (see Figure 6-8) and select **Response file name**. The TradeWinMain program will take the response file as an input argument.

Click **Next**. See 6.2.2, “Response file (properties file)” on page 110 for more information about the response file.

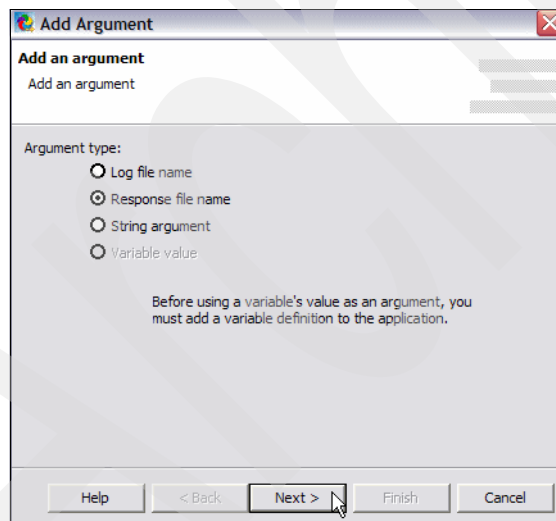


Figure 6-8 Argument pop-up window

- vii. On the next window (as shown in Figure 6-8), enter `Trade.prop` in the Response file name field and click **Finish**.

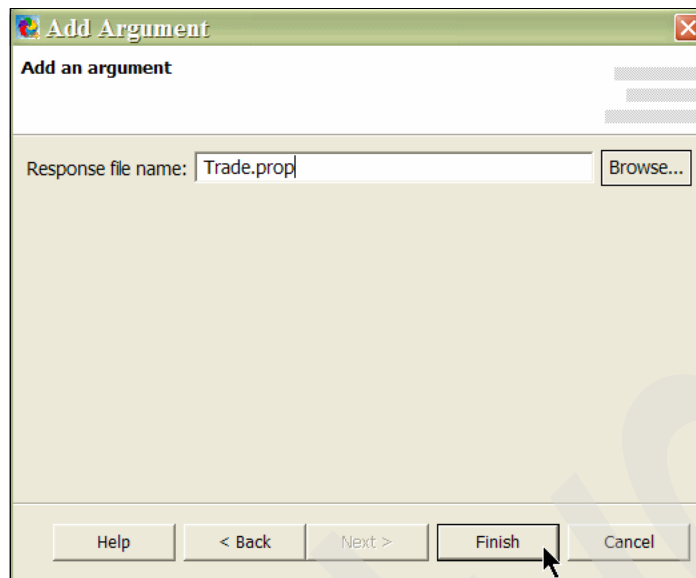


Figure 6-9 Response file name field in adding an argument

Figure 6-10 shows you the completed definition of the Main Program.

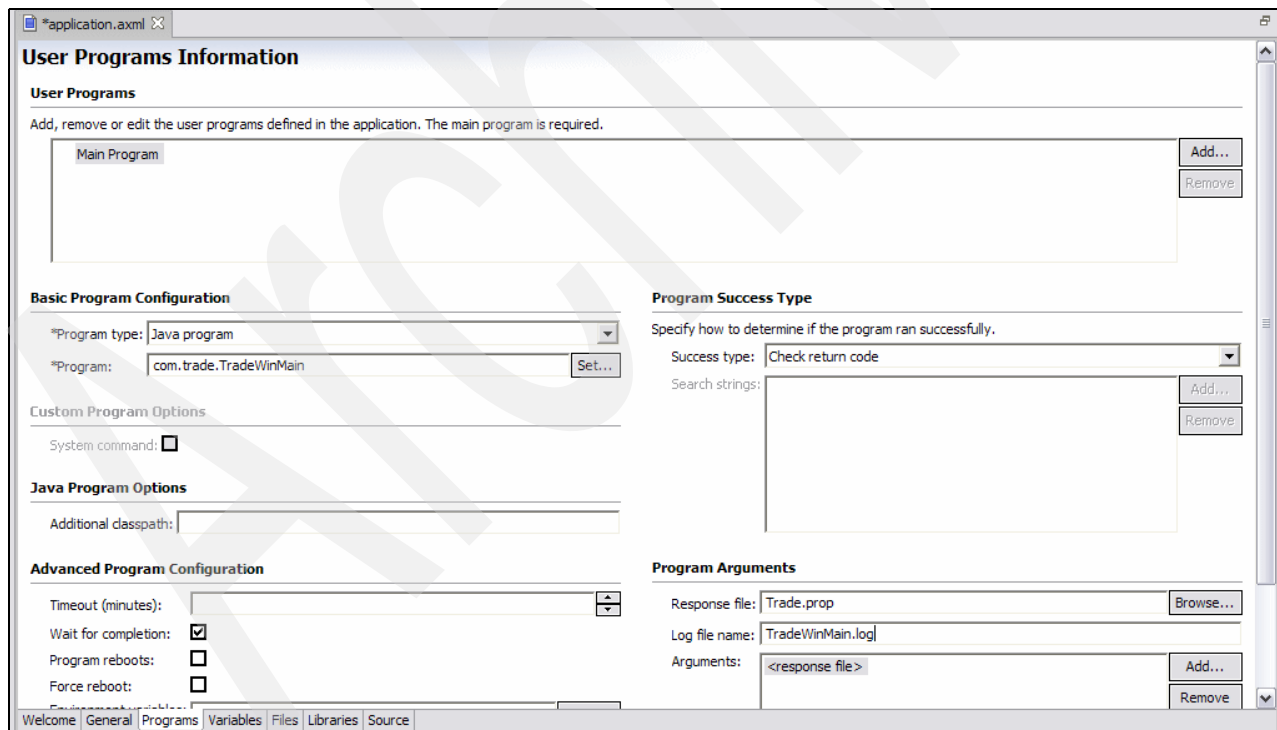


Figure 6-10 Main userProgram definition

- viii. Define the Predeployment Checker program by clicking **Add** in the User Programs section.
- ix. Select the **Predeployment Checker** radio button as shown in Figure 6-11. Click **Next**.

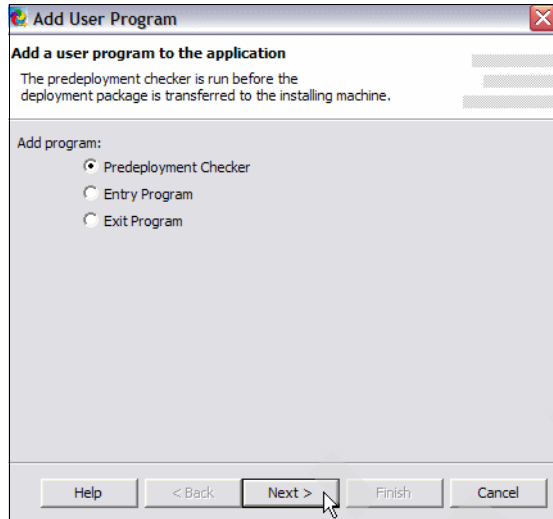


Figure 6-11 Add User Program window

- x. Select **Java Program**. Click **Next** (see Figure 6-12).

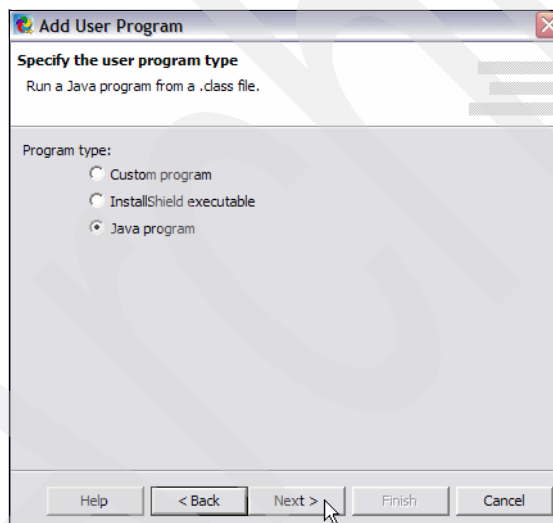


Figure 6-12 Add User Program - Program Type

- xi. Since the Predeployment Checker program does not have any pre-requisite program, leave the next window blank as shown in Figure 6-13. Click **Next**.

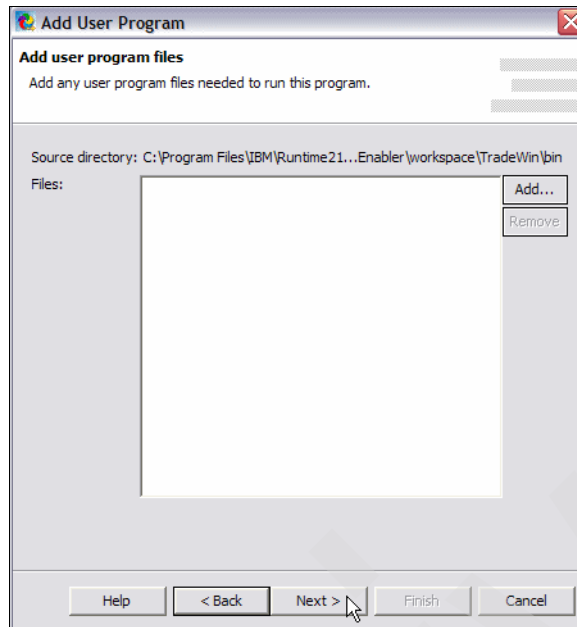


Figure 6-13 Add User Program - Pre-requisite programs to run PDC

- xii. Type in `com.trade.TradeWinPDC` as the Main class. See illustration in Figure 6-14. Click **Finish**.

Note: The TradeWinPDC program does not exist at this time. We will create this in section “Creating Trade6 user programs” on page 113.

The file name `com.trade.TradeWinPDC` is translated with the path `<ER workspace dir>\TradeWin\src\TradeWin\userPrograms\com\ trade\TradeWinPDC`

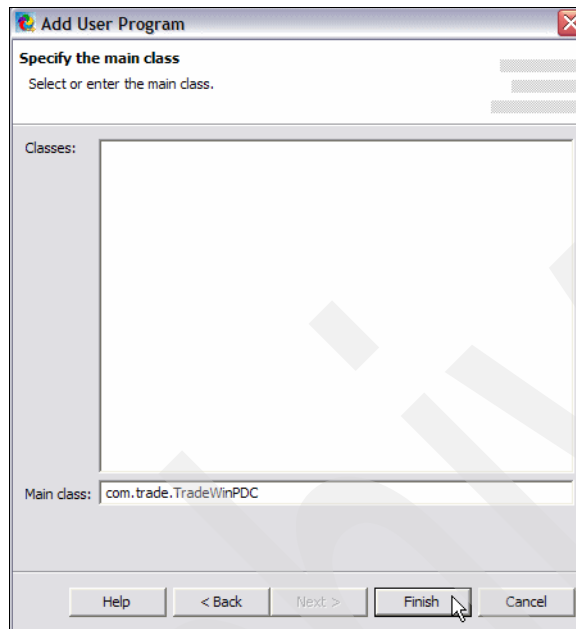


Figure 6-14 Add User Program - Main Class

- xiii. Now that we are back to the User Programs Information window, type in `Trade.prop` as the Response file in the Program Arguments section.
- xiv. Type in `TradeWinPDC.Log` as the Log file name.
- xv. In the Arguments field, click **Add**. In the Add Argument window, select **Response file name**. Click **Next**.
- xvi. On the next window, enter `Trade.prop` in the Response file name field and click **Finish**.

Now the PDC user program is defined (see Figure 6-15). You now have two user programs (Main and PDC) in place as shown in Figure 6-15.

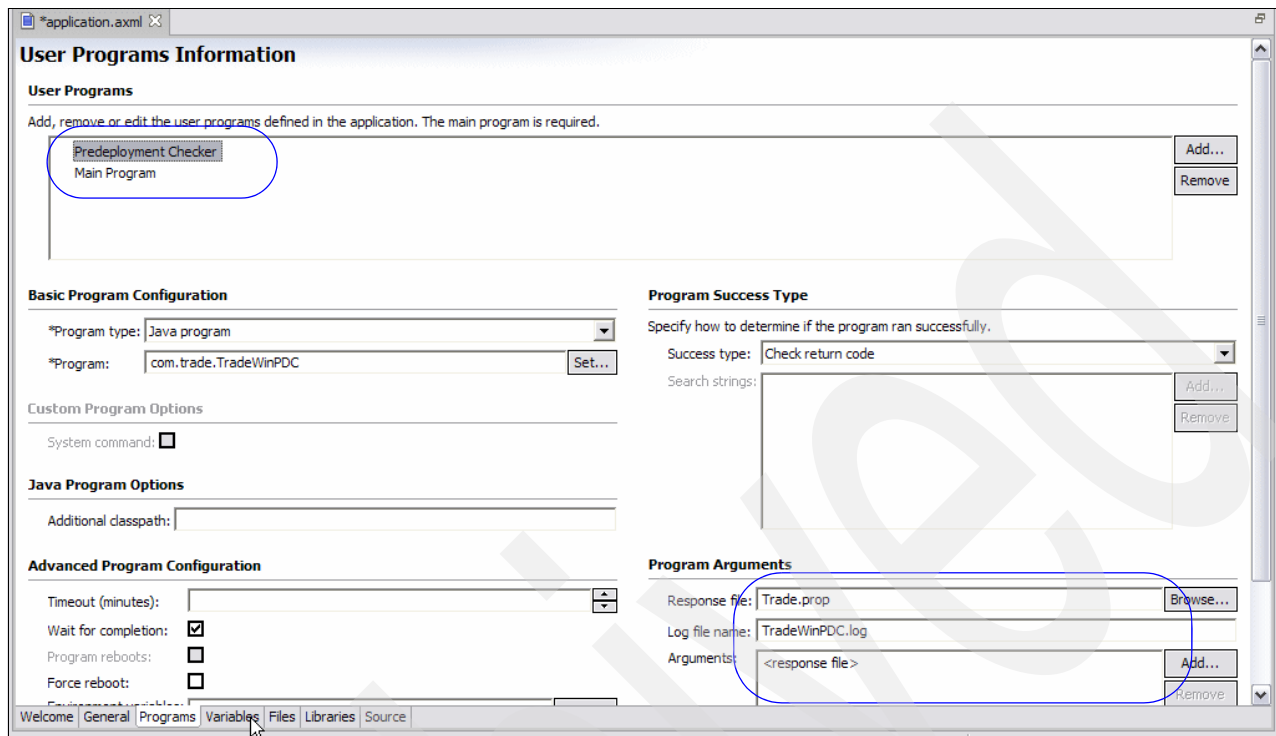


Figure 6-15 User Programs Information - Main and PDC

- c. Click the **Variables** tab. This tab allows you to configure the variables needed in deploying your application. See Figure 6-16.

We need to define two variables that will be exposed to a user during the deployment process: DB2UserID and DB2UserPassword.

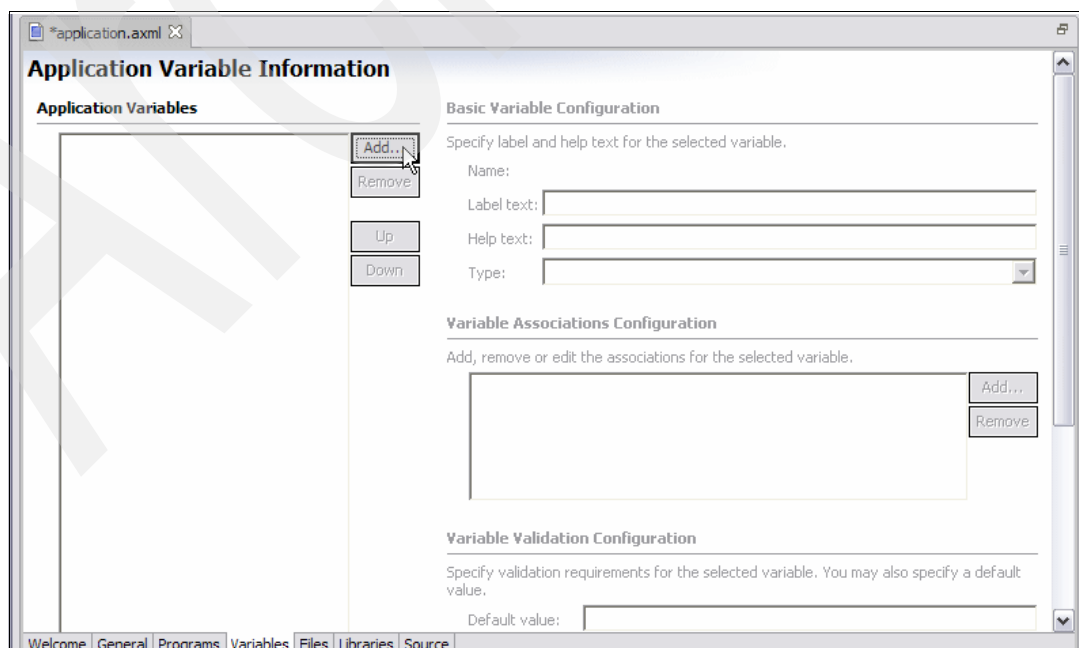


Figure 6-16 Application Variable tab

- i. In the Application Variables section, click **Add**.
- ii. Select **String Variable** as the variable type.
- iii. Type DB2UserID as the variable name.
- iv. Enter DB2 Administrator User ID in the Variable Description field. You should have a pop-up window that look like Figure 6-17. Click **Finish**.

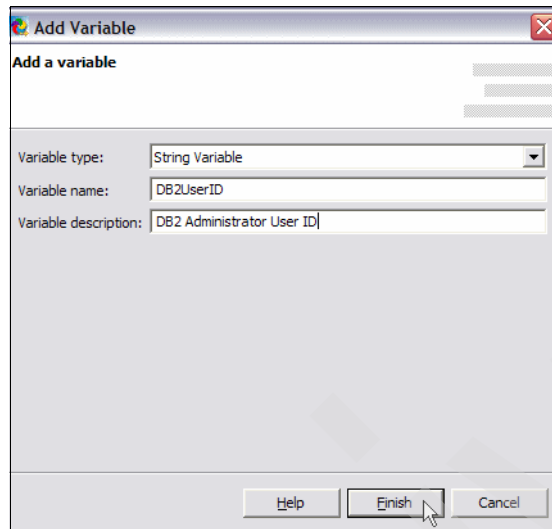


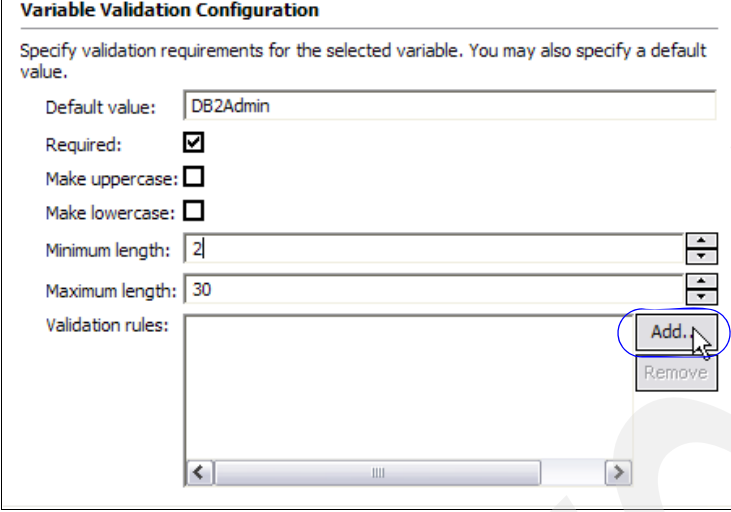
Figure 6-17 Add Variable pop-up window

- v. Define the Variable Validation Configuration for the variable DB2UserId as follows:

- Minimum Length: 2
- Maximum Length: 30
- Validation rules:
 - Valid characters: "@#\$_abcdefghijklmnopqrstuvwxyz0123456789"
 - Invalid values: "ADMINS"
 - Invalid values: "GUESTS"
 - Invalid values: "USERS"
 - Invalid values: "PUBLIC"
 - Invalid values: "LOCAL"
 - Invalid prefix: "IBM"
 - Invalid prefix: "SQL"
 - Invalid prefix: "SYS"
 - Invalid prefix: "_"

Note: Variable validation should depend on the validation rules used by DB2. For instance, refer to DB2 validation rules for the variables DB2UserId and DB2UserPassword.

To add Validation rules, click **Add** on the Validation rules field (Figure 6-18).

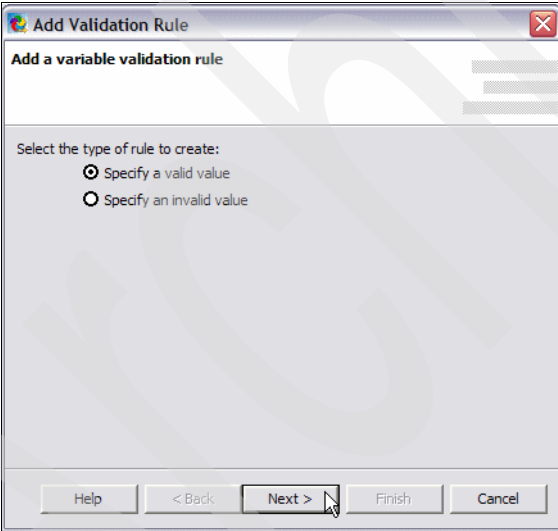


The 'Variable Validation Configuration' dialog box is shown. It has a title bar and a main area with the following fields and controls:

- Default value:** A text box containing 'DB2Admin'.
- Required:** A checkbox that is checked.
- Make uppercase:** An unchecked checkbox.
- Make lowercase:** An unchecked checkbox.
- Minimum length:** A text box containing '2' with a spin button to its right.
- Maximum length:** A text box containing '30' with a spin button to its right.
- Validation rules:** A large empty list box.
- Buttons:** 'Add...' and 'Remove' buttons are located to the right of the 'Validation rules' list box. The 'Add...' button is circled in blue.

Figure 6-18 Add validation rules

In the Add Validation Rule window, select **Specify a valid value** if you need to create an entry for a valid value, or select **Specify an invalid value** for creating an entry for an invalid value. Click **Next** (Figure 6-19).



The 'Add Validation Rule' dialog box is shown. It has a title bar and a main area with the following elements:

- Header:** 'Add a variable validation rule'.
- Select the type of rule to create:** Two radio buttons: 'Specify a valid value' (selected) and 'Specify an invalid value'.
- Buttons:** 'Help', '< Back', 'Next >', 'Finish', and 'Cancel' buttons are at the bottom. The 'Next >' button is highlighted with a mouse cursor.

Figure 6-19 Adding a valid value

In the next window (Figure 6-20), choose the following options accordingly and click **Next**:

- **Specific characters:** Valid characters “@#\$_abcdefghijklmnopqrstuvwxyz0123456789”
- **Complete string:** Invalid characters “ADMINS”, “USERS”, “PUBLIC”, and “LOCAL”
- **Prefix:** Invalid prefix “IBM”, “SQL”, “SYS”, and “_”

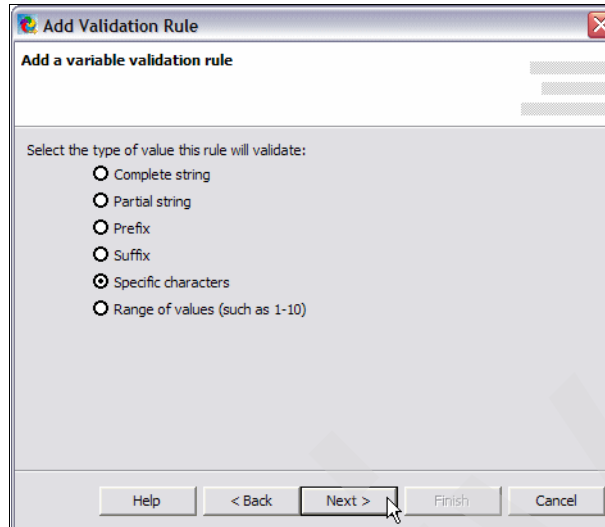


Figure 6-20 Validation type

In the next window (Figure 6-21), enter the appropriate values and click **Finish**. Repeat this process until you have configured all validation rules for this variable.

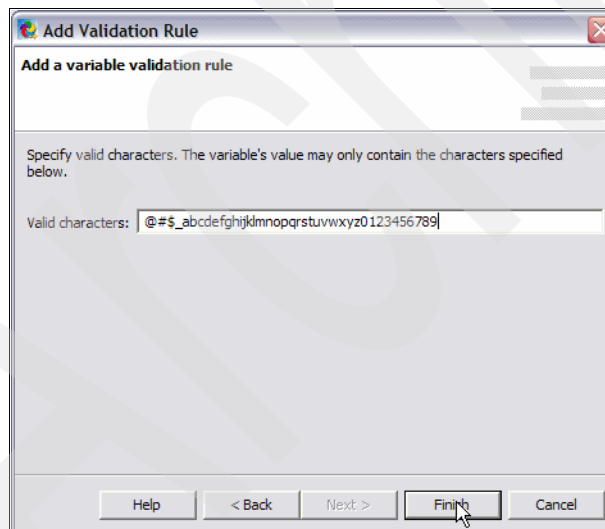


Figure 6-21 Validation value

See Figure 6-22 as an example of a completed variable validation configuration.

Variable Associations Configuration

Add, remove or edit the associations for the selected variable.

Property Association: DB2UserId

Add... Remove

Variable Validation Configuration

Specify validation requirements for the selected variable. You may also specify a default value.

Default value:

Required: ☒

Make uppercase: ☐

Make lowercase: ☐

Minimum length: 2

Maximum length: 30

Validation rules:

- Valid characters: "@#\$_ abcdefghijklmnopqrstuvwxyz0123456789"
- Invalid value: "ADMINS"
- Invalid value: "GUESTS"
- Invalid value: "USERS"
- Invalid value: "PUBLIC"
- Invalid value: "LOCAL"
- Invalid prefix: "IBM"
- Invalid prefix: "SQL"

Add... Remove

Figure 6-22 Variable Validation Configuration

Variable Associations Configuration

After you complete the basic configuration of a variable, configure the variable associations. Variable associations make the variable values available to the user programs by including these values in response files or properties files (see more details in 6.2.2, “Response file (properties file)” on page 110).

There are four types of association:

- ▶ *CID response file association.* The Configuration, Installation, and Distribution response file format supported by DB2 applications.
- ▶ *ISMP response file association.* The response file format for InstallShield MultiPlatform Edition installations.
- ▶ *ISS response file association.* The response file format for InstallShield installations.
- ▶ *Properties association.* Standard properties file format. Java-based user programs can access the value of a variable through this type of association, using the support framework.

Our application wrapper will be responsible for deploying and configuring the Trade6 application. This does not require an InstallShield or CID process, so we will use the simple properties association only. This ensures that the variables we just defined get passed into our user programs through the Trade.prop properties file.

So, we continue with defining the variable associations:

- vi. In the Variable Associations Configuration section, click **Add**.
- vii. In the Add Association dialog, select **Properties Association** (see Figure 6-23). Click **Next**.

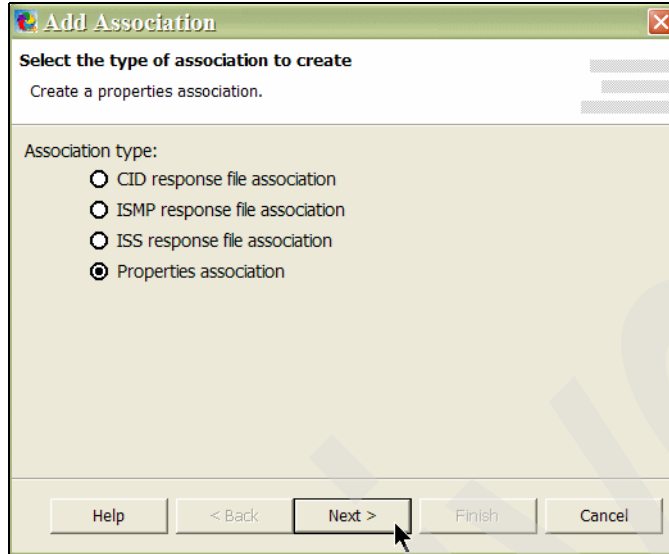


Figure 6-23 Selecting the association type

- viii. Type DB2UserID in the Keyword field (see Figure 6-24). Click **Finish**.

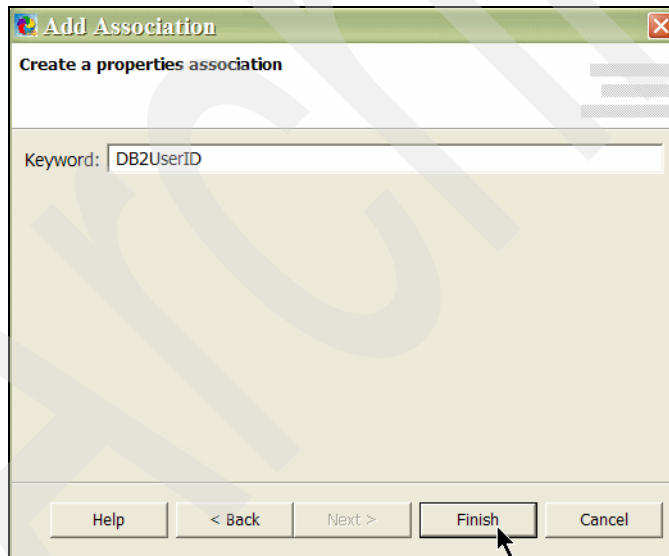


Figure 6-24 Defining the keyword

- ix. If the variable you are adding an association to is a boolean variable, you must configure the *valueIfTrue* and *valueIfFalse* properties of the association. Because the end user can only choose yes or no while configuring a boolean variable, the application developer must specify the values to be inserted into the properties or response file. To specify these values right-click the variable association, select **Properties**, then edit the *valueIfTrue* and *valueIfFalse* properties.

- x. Add another variable by clicking **Add** on the Application Variables section.
- xi. In the Variable type field, select **Password Variable**.
- xii. Enter DB2UserPassword in the Variable name field.
- xiii. Enter DB2 Administrator Password in the Variable description field.
- xiv. Define the Variable Validation Configuration for DB2UserPassword as follows:
 - Minimum Length: 6
 - Maximum Length: 127
 - Validation rules:
 - Valid characters: "@#\$_abcdefghijklmnopqrstuvwxyz0123456789"
- Refer to Figure 6-22 for an example of a completed variable validation configuration.
- xv. Create variable association for DB2UserPassword.
- See the completed tab in Figure 6-25.

Application Variable Information

Application Variables

String Variable: DB2UserId Add...
 Password Variable: DB2UserPassword Remove
 Up
 Down

Basic Variable Configuration

Specify label and help text for the selected variable.

*Name: DB2UserPassword
 *Label text: DB2 Administrator Password
 Help text: The Administrator Password used to connect to DB2.
 Type: Typical variable

Variable Associations Configuration

Add, remove or edit the associations for the selected variable.

Property Association: DB2UserPassword Add...
 Remove

Variable Validation Configuration

Specify validation requirements for the selected variable. You may also specify a default value.

Default value:
 Required: ☒
 Make uppercase: ☐
 Make lowercase: ☐
 Minimum length: 2
 Maximum length: 30
 Validation rules: Valid characters: "@#\$_abcdefghijklmnopqrstuvwxyz0123456789"
 Invalid value: "ADMINS"
 Invalid value: "GUESTS"
 Invalid value: "USERS"
 Invalid value: "PUBLIC"
 Invalid value: "LOCAL"

Welcome General Programs **Variables** Files Libraries Source

Figure 6-25 The Variables tab

- d. Save your application project by selecting **File** → **Save** from the main menu.

6.2.2 Response file (properties file)

A response file (or properties file) contains parameters used by your user program. At runtime a user program reads in the parameters' values. By modifying the values of the parameters in the response file, a user can effectively change the way this user program executes. If you also use the same response file for several user programs, you can share the values of the parameters between multiple user programs.

The sample response file is shown in Example 6-1 on page 111. It has the format:

```
<property name> = <value>
```

Some of the properties don't have values. In most of the cases, these properties are associated with the application variables (see "Variable Associations Configuration" on page 107). At runtime, a user provides the values for these variables in Deployment Wizard. Even if a property has a value, either a user (through the variable association) or a user program can change it at runtime. However, this doesn't change the original response file, just a temporary runtime copy.

In some cases the user program also passes the response file to an installation procedure that it might call. For example, it may pass on a response file to an InstallShield program or to a script that performs some deployment and configuration tasks. In the case of Trade, you see later that we do this by passing the response file from the user program to the wsadmin JACL script that we use.

The Trade.prop properties file

The Trade properties file defines the information needed to install the Trade6 application. To create this file, use the following steps:

1. From Package Explorer, expand the folders **TradeWin** → **src**.
2. Right-click **TradeWin** folder and select **New** → **File** as shown in Figure 6-26.

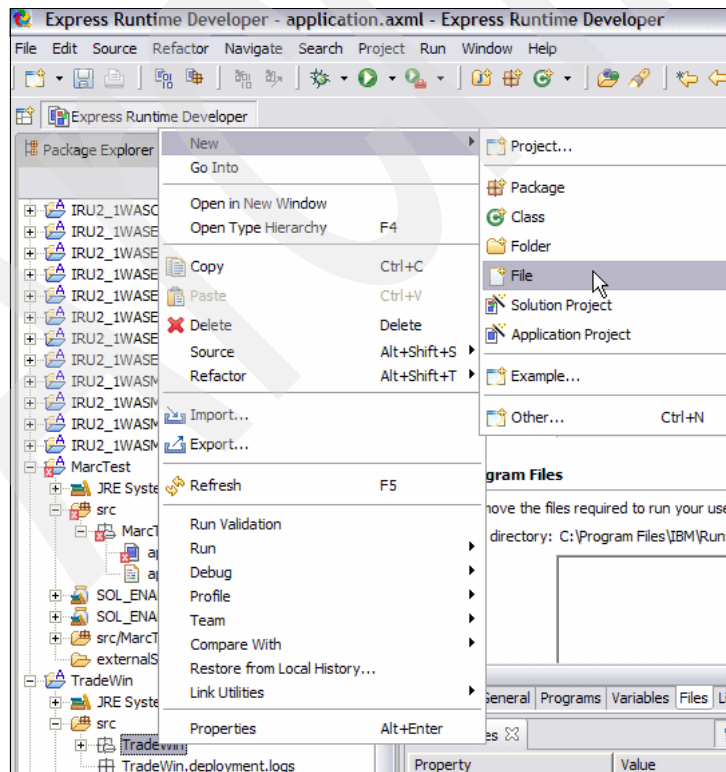


Figure 6-26 Creating a new file using the Package Explorer

3. In the New File window, enter `Trade.prop` on the File name field (Figure 6-27). Click **Finish**.

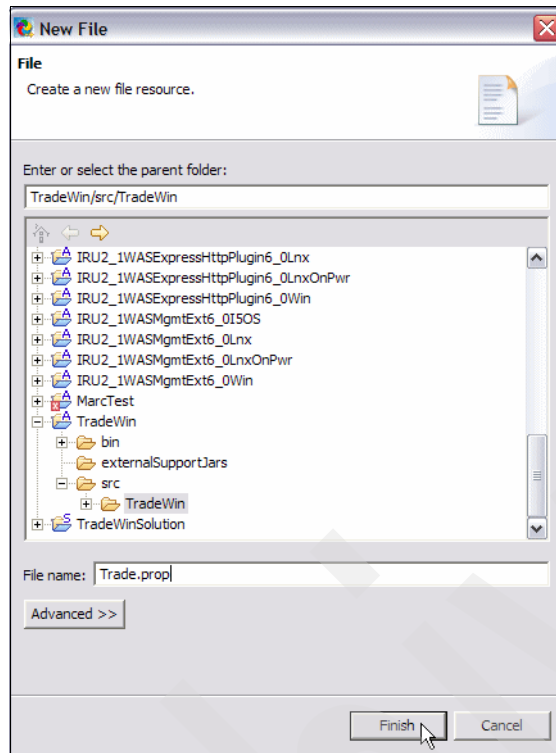


Figure 6-27 New `Trade.prop` file

The `Trade.prop` content is created manually. Just double-click this file in Express Runtime Developer. To illustrate how the `Trade.prop` file should look, see Example 6-1.

Example 6-1 Trade properties file

```
#-----
#
# Trade Properties File
#
# HINTS:
#   - Set path names using shortname, e.g., "Program Files" is "progra~1"
#   - Use forward slash in directory entries
#
# Note that some of these values are set in Main
#-----

# The name of the WebSphere configuration script to be run
TRADE.WASscript=WebSphereScript.jacl

# The name of the DB2 script to be run
TRADE.DB2script=DB2Script.bat

# The scripts are part of the userPrograms fileList in the application.xml.
# The scriptsDir is the directory under which the scripts were placed, as
# specified in the application.xml.
TRADE.scriptsDir=Trade_ScriptsDir

# Variables used for WAS
WAS.version=6.0.0.0
```

```

WAS.serverName=server1
WAS.profile=default
# WAS installable apps directory is set in the Main
WAS.installableAppsDir=

# -----
# Variables used for WAS configuration steps
# -----

# The name of the application to be installed
WAS.appName=Trade

# the install EAR or WAR file
WAS.appInstallFile=trade.ear

# the contextRoot -- used if the application is installed using a war file
WAS.contextRoot=trade

# the JDBC classpath will be set in the Main --
# c:/progra~1/IBM/SQLLIB/java/db2java.zip
WAS.JDBCClasspath=

# the JDBCProvider name to be created
WAS.JDBCProviderName=TradeDB2Provider

# the DataSource to be created
WAS.datasourceName=TradeDataSource
WAS.datasourceDesc="This datasource is used by the Trade Application"
WAS.datasourceCategory="Trade Application"

# the J2C Authorization entry to be created
WAS.authAliasName=TradeDataSourceAuthData
WAS.authAliasDesc="JAASDataAuth for the Trade database"

# the J2C Authorization entry to be created for JMS resource
WAS.JMSauthAliasName=TradeOSUserIDAAuthData
WAS.JMSauthAliasDesc="JAASDataAuth for the Trade JMS resources"

# the Local OS userid and password
WAS.DefaultOSUser=LocalOSUserID
WAS.DefaultOSPasswd=password


# the DB2 database name and instance
DB2.databaseName=tradedb
DB2.db2instance=DB2
# the DB2 Administrator userid and password
DB2.AdminID=
DB2.AdminPW=

# The name of the IHS service, in quotes, to be started using the net command,
# e.g., net start "IBM HTTP Server 6.0"
# If a web server service has not been defined on the system, do not provide a value
IHS.webServerService="IBM HTTP Server 6.0"

```

Tip: The choice of what values to use in a response file is determined by what values are required by your user program and any installation procedure it may invoke. For the Trade6 application, the user program will invoke a wsadmin script for its installation procedure. As another example, you can look at the user programs for the supplied middleware and see that these user programs often invoke InstallShield installation procedures. Your own application wrapper user program could do either of these.

If you are invoking a product's InstallShield procedure, then you might want to begin creating your response file based upon a typical response file that the InstallShield procedure requires. If you are invoking your own script, as we do for Trade, then you can define the response file contents to match what your script requires.

6.2.3 Creating Trade6 user programs

The next task in developing our wrapper for Trade6 is the creation of user programs. In the previous section we defined the following programs:

- ▶ Main Program: TradeWinMain
- ▶ Predeployment Program: TradeWinPDC

Basically, we use Java to write these programs and place them in *<ER workspace dir>\TradeWin\src\TradeWin\userPrograms\com\trade*.

We also use the APIs in the Support Framework package to perform functions commonly needed during installation.

Note: To view the API documentation of the Support Framework package, open the index.html file in the path *<ER workspace dir>\Runtime21\SolutionEnabler\Support_Framework*.

Since the program codes are a bit too long for this chapter's discussion, we only look at some of the significant snippets. You may refer to Appendix A, "Source code for Trade6 user programs and script files on Windows" on page 257, to see the complete source listing of all the user programs and script files we use on Trade6. Alternatively, you can download the wrapper's source code as described in Appendix D, "Additional material" on page 435.

Tips for developing user programs

Express Runtime Developer comes with many sample applications and wrappers. Most of them are the middleware images and the corresponding wrappers. It is strongly recommended to use these wrappers as is. However, you need to create a custom wrapper for any of your own applications.

To simplify the creation of the custom wrappers, we recommend to use the sample wrappers as good programming examples. In our book we model the Trade6 wrapper after the wrapper in the IRU2_1SampleWin project (see Figure 6-28).

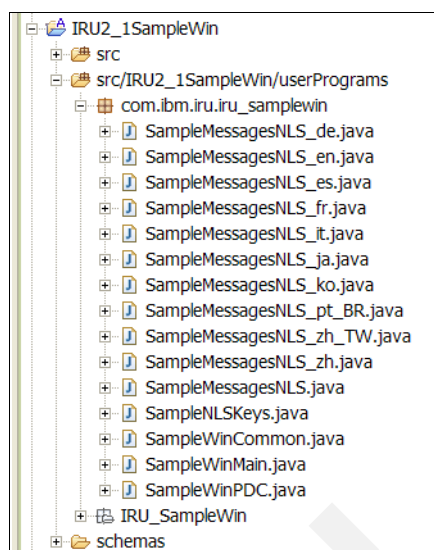


Figure 6-28 Sample project

The key Java classes in this application project are:

- ▶ **SampleWinMain.java.** This is the Main user program. It performs most of the installation steps.
- ▶ **SampleWinPDC.java.** This is the predeployment checker program. It verifies the pre-requisites before transferring the application files and installing them on a target system.
- ▶ **SampleMessageNLS.java and SampleMessageNLS_xx(_xx).java.** These files defines the messages generated by the user programs. If you plan to support multiple languages, you need to translate all the messages in each language and store them in the language specific file.
- ▶ **SampleNLSKeys.java.** This file contains the statics references to the resource bundles.

In many cases we use copy-and-paste technique (with some modifications) to create the Trade6 user programs, because most of the source code in the sample wrapper matches the Trade6 wrapper functionality.

In addition, the sample wrappers contain good examples of using the Support Framework API (see 5.3, “Support Framework API” on page 86).

The TradeWinPDC.java program

The TradeWinPDC.java program represents our Predeployment Checker that determines the conditions for deployment. If the state for deployment is not right, then the application image is not transferred over the network to the target system, which avoids wasting time and network traffic. The program performs the following checks:

- ▶ Determine if WAS is installed
- ▶ Determine if DB2 is installed
- ▶ Check that Trade6 is not installed already

The TradeWinPDC.java program uses a wsadmin script written in Jacl to determine if these components exist on the target system. If this returns true, the program does not re-install the application and can be set to report an error and either stop or continue the deployment. You may create or copy the codes to perform these checks from the built-in projects provided in Express Runtime Developer.

During the installation check, TradeWinPDC will return one of the integer constants that are defined in the *SupportBase* class:

- ▶ PDC_EXISTS — This value is returned when PDC detects that the component already installed.
- ▶ PDC_DOES_NOT_EXIST — This value is returned when PDC detects that a component is not installed on the target system.
- ▶ FAILURE — This is an indication of an error situation. It can mean that the result file does not exist, or there is some other unexpected error.

Example 6-2 shows the part of TradeWinPDC.java program that checks if the application exists on the target system.

Example 6-2 The TradeWinPDC: Checking for the existence of Trade6 via wsadmin

```
// Determine if the application is already installed
// Run the WebSphere Admin application (wsadmin), the
// -f option indicates a jacl script is the next arg
// After the script name, the arg for the jacl script is:
// - the application name
String checkCommand = "cmd /C " + getWasProfileBinDir() + "wsadmin -f " +
    ivHelper.getUnpackedDir(this).replace('\\', '/') + getScriptsDir() +
    TradeWinCommon.CHECK_INSTALL_SCRIPT + // Script to run
    " " + getAppName(); // the application name

// invoke and log the command
rc = invokeCommand(checkCommand, checkCommand);
```

This program uses two other Java programs for message translations: TradeMessagesNLS.java and TradeMessagesNLS_en.java.

To create the Predeployment user program class using Express Runtime Developer, follow these steps:

1. Start Express Runtime Developer by selecting **Start** → **Programs** → **IBM Express Runtime 2.1** → **Express Runtime Developer**.
2. Expand the **TradeWin** folder.
3. Right-click the folder **src/TradeWin/userPrograms** and select **New** → **Class** (see Figure 6-29).

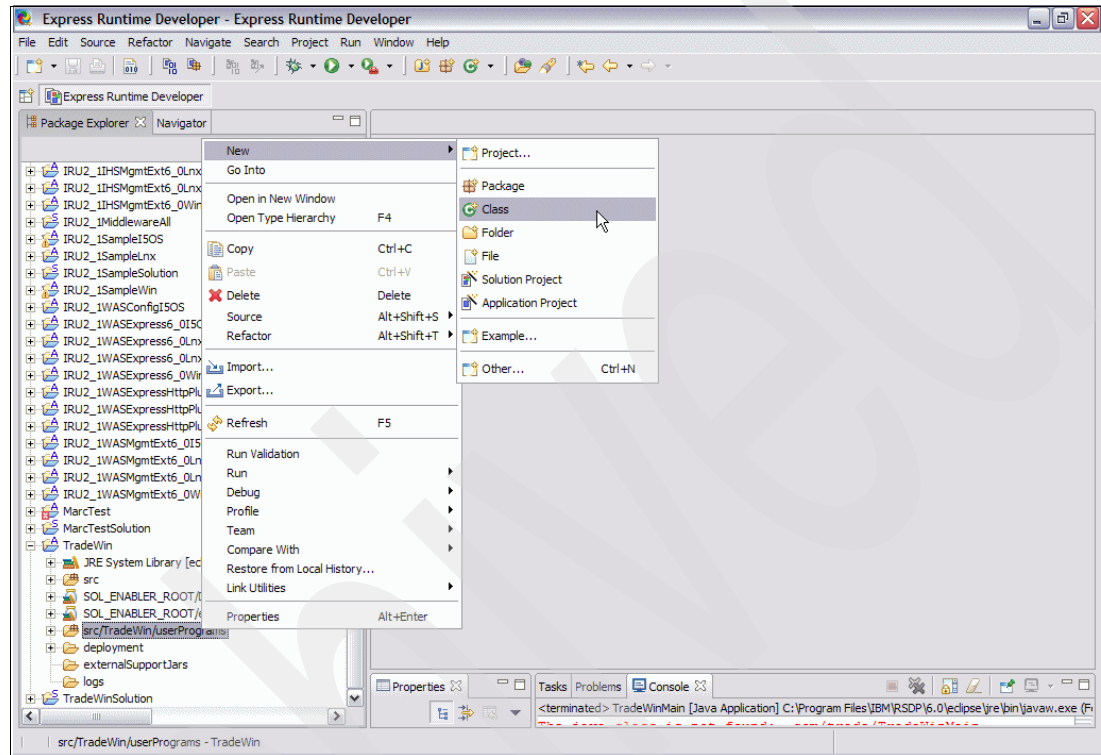


Figure 6-29 Creating a new class

4. In the New Java Class window, enter or select the following values (Figure 6-30):
 - a. Package: `com.trade`
 - b. Name: `TradeWinPDC`
 - c. Superclass: `com.ibm.jsdt.support.SupportWindowsBase`. The superclass you choose does not have to be the `SupportWindowsBase`, but by choosing this, you will inherit methods from the Support Framework that will make your user program easier to write.
 - d. Which methods stubs would you like to create: **`public static void main(String[] args)`**
 - e. Click **Finish**.

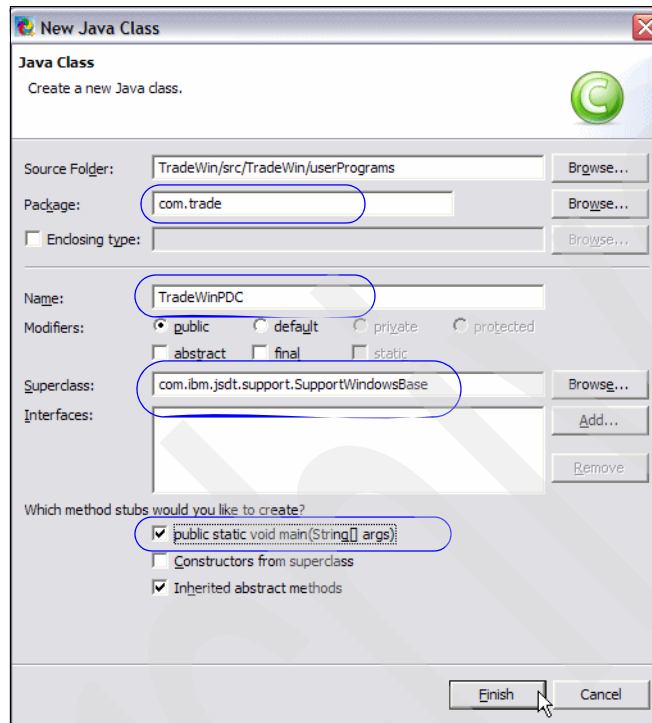


Figure 6-30 New TradeWinPDC Java

5. The class opens in the Java editor (see Figure 6-31). You may refer or copy the TradeWinPDC source code listing in Appendix A, “Source code for Trade6 user programs and script files on Windows” on page 257, to create this program.

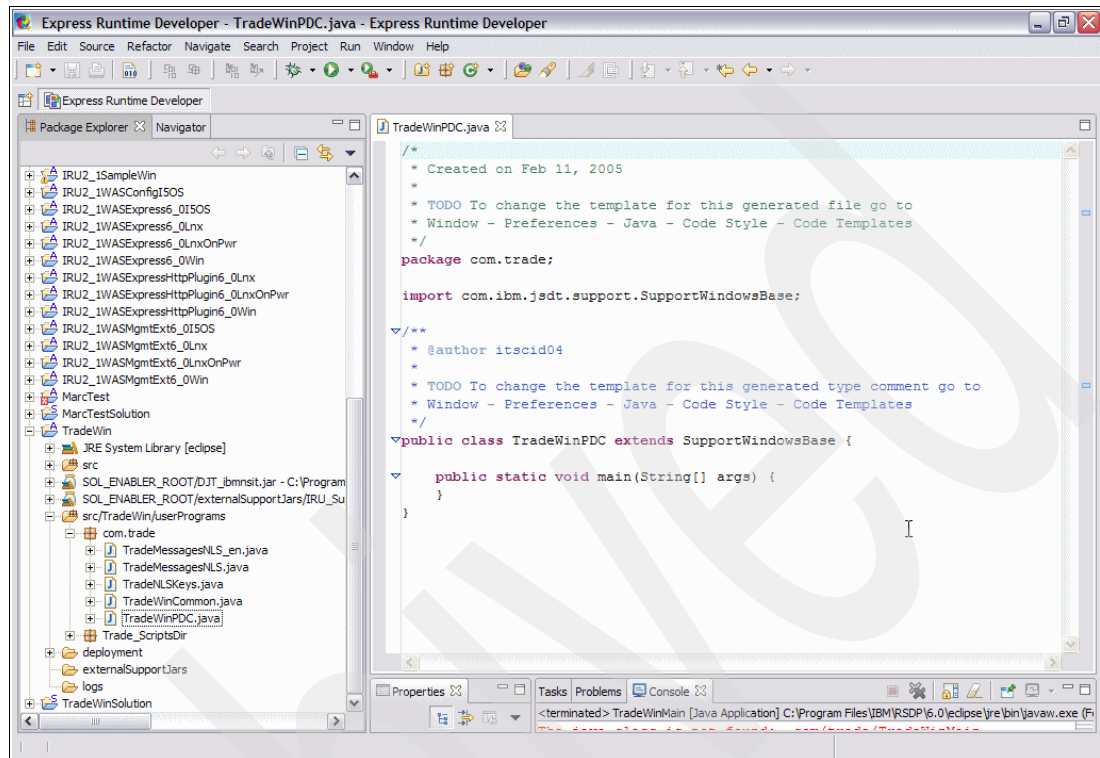


Figure 6-31 TradeWinPDC programming workspace

The TradeWinMain.java program

The TradeWinMain.java program is the program that performs the installation of Trade6. These are the tasks performed by the main program:

- ▶ Retrieves the values set in the initial response file
- ▶ Reads the values set by a user in Deployment Wizard
- ▶ Determines installation location of WAS and DB2
- ▶ Copies the application file (EAR file) to the target system
- ▶ Invokes the DB/2 script to setup the application's database and tables
- ▶ Runs the WAS script to configure the WAS environment for the application and deploys Trade6
- ▶ Generates the HTTP plug-in
- ▶ Enables the plug-in by restarting the HTTP Server

As an example, let us take a look at the copy operation.

It copies the Trade.ear file into the WAS installableApps directory (for instance: C:\Program Files\IBM\WebSphere\AppServer\installableApps).

Example 6-3 shows the snippet of code that performs this function.

Example 6-3 Snippet of code in TradeWinMain.java to copy Trade.ear

```
private int copyApp() {
    int rc = FAILURE;

    // Copy the ear or war file to the WAS installableApps directory
    setSource(ivHelper.getUnpackedDir(this) + getAppFile());
    setTarget(getInstallableAppsDir());

    if (ivHelper.fileCopy(this)) {
        .....
    }
```

The ivHelper object in the example is of type *SupportWindowsHelper*. This class is part of the Support Framework API. The invocation of the *getUnpackedDir* method returns the path to the directory where the application file will be unpacked (copied).

The main program access the properties in the response file (Trade.prop) using the following pattern:

```
setKey(SOME_KEY_VALUE);
ivHelper.getResponseFileValue(this);
```

The *setKey()* and *getResponseFile()* methods are used in TradeWinMain as shown in Example 6-4.

Example 6-4 The getResponseFileValue

```
setKey(TradeWinCommon.APP_FILE_KEY); // the ear or war file to be installed
setAppFile(ivHelper.getResponseFileValue(this));
setKey(TradeWinCommon.DB2_SCRIPT_KEY); // DB2 script
setDB2Script(ivHelper.getResponseFileValue(this));
setKey(TradeWinCommon.SCRIPTS_DIR_KEY); // the directory within userPrograms where the
scripts are located
setScriptsDir(ivHelper.getResponseFileValue(this));
setKey(TradeWinCommon.DB_NAME_KEY); // the name of the database
setDbName(ivHelper.getResponseFileValue(this));
setKey(TradeWinCommon.WAS_SCRIPT_KEY); // WAS script
setWasScript(ivHelper.getResponseFileValue(this));
setKey(TradeWinCommon.IHS_SERVER_NAME_KEY); // the name of the IHS server
setIhsServerName(ivHelper.getResponseFileValue(this));
```

The TradeWinCommon class contains the constants that define the property names. A snippet of this file shown in Example 6-5.

Example 6-5 Defining property names

```
public static final String DB2_SCRIPT_KEY = "TRADE.DB2script";
public static final String SCRIPTS_DIR_KEY = "TRADE.scriptsDir";
public static final String DB_NAME_KEY = "DB2.databaseName";
public static final String DB2_ADMIN_ID_KEY = "DB2.AdminID";
public static final String DB2_ADMIN_PW_KEY = "DB2.AdminPW";
public static final String JDBC_CLASSPATH_KEY = "WAS.JDBCclasspath";
public static final String WAS_INSTALLABLE_APPS_DIR_KEY = "WAS.installableAppsDir";
public static final String APP_FILE_KEY = "WAS.appInstallFile";
```

If you compare the values of these constants to the Trade.prop file (see Example 6-1 on page 111), you notice that these are the property names.

As a result, the call to the `setKey` method sets the name of the property. And the call to one of the `set<property name>()` methods (for example, `setAppFile`) sets this property value for usage within this class (user program).

`Trade.prop` contains pre-defined values. However, these can be overwritten by a deployer or user program at deployment time using the variables in `application.xml` as follows:

- ▶ Define a variable. This allows the deployer to enter a value for it at deployment time.
- ▶ Define an association between the variables and the response file. This causes the Deployment Wizard to automatically update the response file with whatever value the deployer enters for the variable.

As a result, if a variable is associated with the response file, you can access its value by using `getResponseFileValue()` method.

In some cases, using an association to update the response file may not be possible. Remember that the response file will be passed first to your user program, and then the user program will typically pass it on to some installation procedure. Take the example of a user program that invokes an InstallShield installation procedure and consider the following discussion. Most InstallShield silent installers will be strict about what are valid input values and so will not tolerate any unrecognized values. This would be a problem if you were to define an association for a variable that was only intended for use within your user program, because it would also get passed to the InstallShield routine, causing it to fail.

To let the user programs to pass some values between each other without using a response file, Express Runtime provides the following association type - *properties association* (see more information in “Variable Associations Configuration” on page 107). A special file, called *IBMNSI.properties*, is used to support this type of association. This file is maintained by Express Runtime. To access variables that are defined with the properties associations (through the *IBMNSI.properties* file) a user needs to use the special API, which is defined in the `SupportHelper` class (part of the Support Framework API).

This is the example of using this API:

```
setVariableName(SOME_VARIABLE_NAME);
ivHelper.getIbmNsiPropValue(this);
```

Example 6-6 shows a good sample from the `Trade6` user program to retrieve the values of two variables: `DB2UserID` and `DB2UserPassword`. Likewise, `setDB2UserID` and `setDB2PassWord` are two methods to set the values of two variables in the current object.

Example 6-6 TradeWinMain: Getting properties from support framework

```
setVariableName("DB2UserId");
setDB2UserId(ivHelper.getIbmNsiPropValue(this));
setVariableName("DB2UserPassword");
setDB2PassWord(ivHelper.getIbmNsiPropValue(this));
```

As we define a command to execute on the target system, we access the properties that have been read from the response file. Example 6-7 demonstrates how we create a command to run the DB2 script (from the `runDB2Script()` method).

Example 6-7 TradeWinMain: Running DB2 script

```
// Run batch file
// db2cmd command options:
// -c is to execute the command and then terminate
// -i is to share same console (window)
// -w is to wait until the cmd.exe process ends
// The name of the script is then followed by the args used by the script
// - the database name
// - the full path to the script dir
// - the full path to the documents dir
// - the db2 user
// - the db2 password

String command = "db2cmd -c -i -w " + fullScriptsDir + script + // Script to run
    " " + getDbName() + // database name
    " " + fullScriptsDir + // Scripts directory
    " " + getDB2UserId() + // DB2 userid
    " " + getDB2PassWord(); // DB2 password

rc = invokeCommand(command);
```

The main program also runs the .jacl script to configure the WebSphere environment and installation properties of the solution. To obtain the WebSphere script name, DCMGMT.WASScript is read in the properties file. As an example, the following code in TradeWinMain.java shows how to run the WebSphere script (see Example 6-8).

Example 6-8 TradeWinMain: Running the WebSphereScript()

```
// Run the WebSphere Admin application (wsadmin)
// The -f option indicates a jacl script is the next arg
// After the script name, the args for the jacl script are:
// - directory where the scripts are located
// - the fully qualified properties file name
// - the DB2 administrator userID
// - the DB2 administrator password
String command = "cmd /C " +
    shortWasProfileBinDir +
    "wsadmin -f " +
    fullScriptsDir + getWasScript() + // Main jacl script
    " " + fullScriptsDir + // the scripts directory
    " " + getResponseFileName().replace('\\', '/'); // and the properties file
rc = invokeCommand(command, command);
```

The WAS GenPluginCfg batch file is used to generate the HTTP plug-in. Example 6-9 shows how the script is called on line 153.

Example 6-9 TradeWinMain: generateHTTPPlugin()

```
private int generateHTTPPlugin() {
    int rc = SUCCESS;

    // Get the Windows short path for the WAS bin directory
    setPath(getWasProfileBinDir());
    String shortWasProfileBinDir = ivHelper.getWindowsShortPath(this);

    // Use the WAS GenPluginCfg to generate the HTTP plug-in config file
    String command = "cmd /C " + shortWasProfileBinDir + "GenPluginCfg.bat";
    rc = invokeCommand(command);

    return rc;
}
```

Once the HTTP plug-in has been generated, the HTTP Server needs to be restarted to pick up the new values from the plug-in configuration file. See the function `restartHTTPServer()` in Example 6-10. The `getIhsServerName()` method is used to the name of the HTTP server instance.

Example 6-10 TradeWinMain: restartHTTPServer()

```
private int restartHTTPServer() {
    int rc = SUCCESS;

    // use the net command to stop the server, log an error but continue if an error
    occurs
    String command = "cmd /C net stop " + getIhsServerName();
    int ignoredRC = invokeCommand(command);
    if (ignoredRC != SUCCESS) {
        // log an error if not successful
        setMessage(getResourceString(TradeNLSKeys.STOP_HTTPSERVER_FAIL));
        ivHelper.log(this);
    }

    // use the net command to start the server
    command = "cmd /C net start " + getIhsServerName();
    rc = invokeCommand(command);

    if (rc != SUCCESS) {
        // log an error if not successful
        setMessage(getResourceString(TradeNLSKeys.START_HTTPSERVER_FAIL));
        ivHelper.log(this);
    }

    return rc;
}
```

To create the Main user program class using Express Runtime Developer, follow these steps:

1. Start Express Runtime Developer by selecting **Start → Programs → IBM Express Runtime 2.1 → Express Runtime Developer**.
2. Expand the **TradeWin** folder.
3. Right-click the folder **src/TradeWin/userPrograms** and select **New → Class**.

4. In the New Java Class window, enter or select the following values (Figure 6-32):
- Package: `com.trade`
 - Name: `TradeWinMain`
 - Superclass: `com.ibm.jsdt.support.SupportWindowsBase`
 - Which methods stubs would you like to create: **public static void main(String[] args)**
 - Click **Finish**.

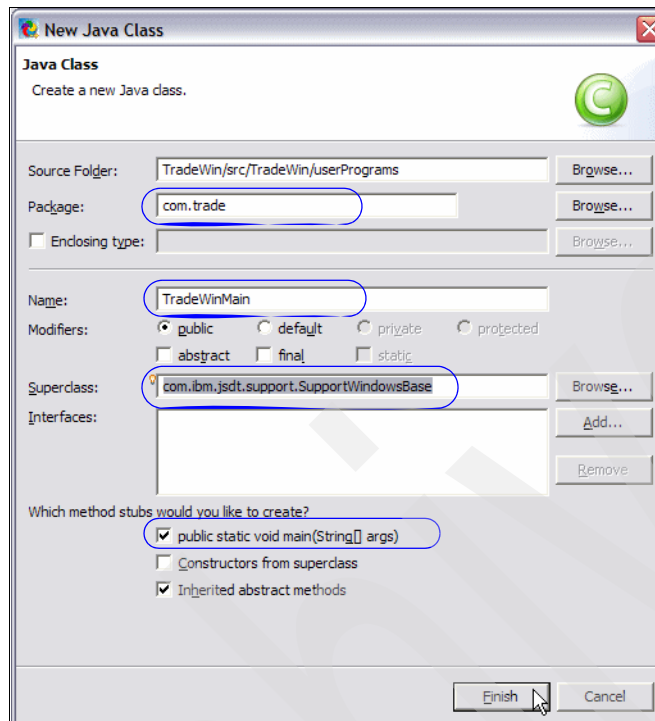


Figure 6-32 New TradeWinMain java

5. The class opens in the Java editor (see Figure 6-31). You may refer or copy the TradeWinMain source code listing in Appendix A, “Source code for Trade6 user programs and script files on Windows” on page 257, to create this program.

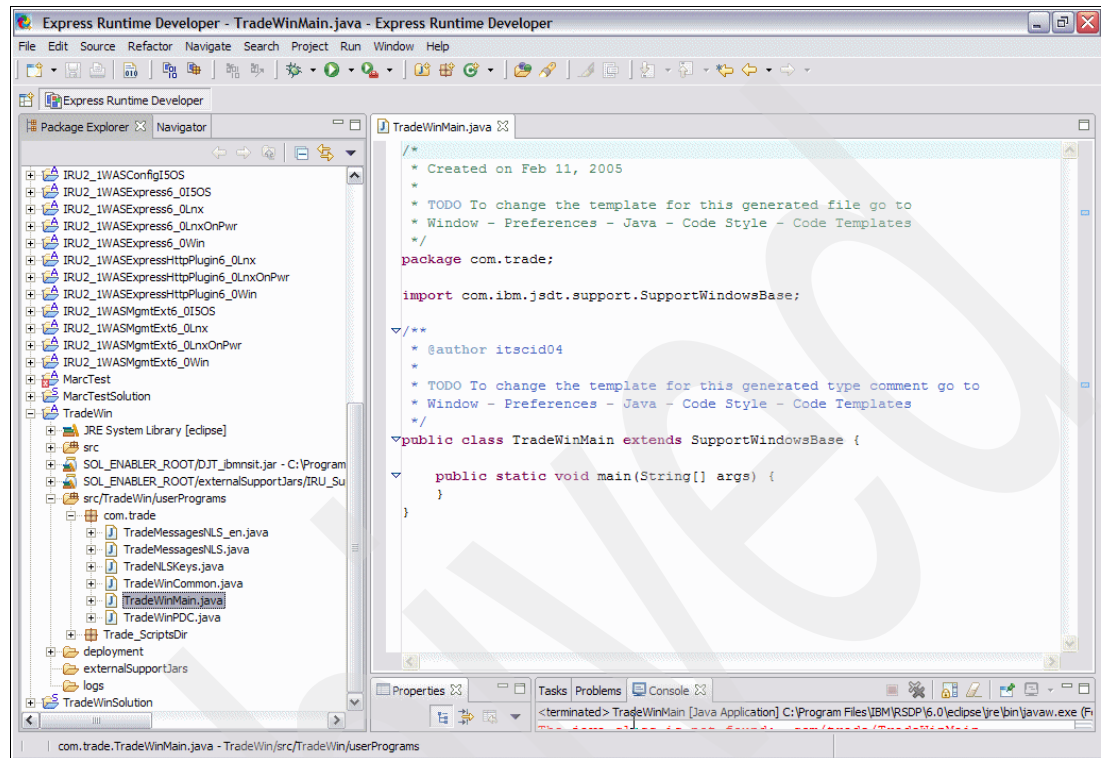


Figure 6-33 TradeWinMain programming workspace

Compiling user programs

Once the Java classes are coded, class files should be generated. Express Runtime Developer comes with the Java Builder that builds Java programs using a compiler that implements the Java Language Specification. The Java Builder can build programs incrementally as individual Java files are saved.

Problems detected by the compiler are classified as either warnings or errors. The existence of a warning does not affect the execution of the program; the code executes as if it were written correctly. Compile-time errors (as specified by the Java Language Specification) are always reported as errors by the Java compiler. For some other types of problems you can, however, specify if you want the Java compiler to report them as warnings or errors, or to ignore them. To change the default settings, use the preference page from the menu bar: **Window** → **Preference** → **Java** → **Compiler**.

There are two ways to trigger a build command:

- ▶ Building automatically: If auto build is turned on (from the Express Runtime Developer menu, **Project** → **Build Automatically**), then an incremental build occurs every time you save a modified workbench resource.
- ▶ Building manually: You can perform a manual build using a keyboard shortcut (**Ctrl+B**), a project's pop-up menu, or the Project menu in the menu bar (**Project** → **Build All**).

The default selection for compiling Java programs is automatic. Should there be a need to compile the user programs manually, use Express Runtime Developer. To show how this is done, highlight all the user programs created and select **Project** → **Build All** from the menu bar (Figure 6-34).

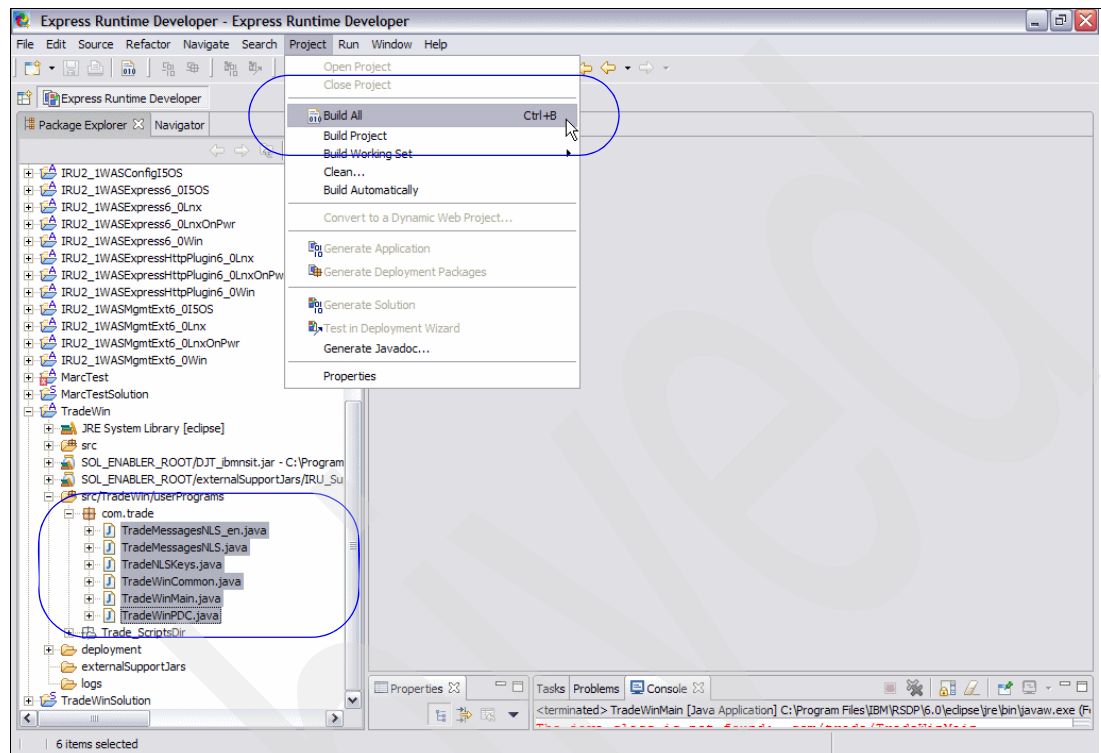


Figure 6-34 Building all Java programs

This creates the class files in the default build path *ER install dir\TradeWin\bin\com\trade* (see Figure 6-35). These class files are then included in the application project as we go to 6.2.4, “Completing the Trade6 application project” on page 127.

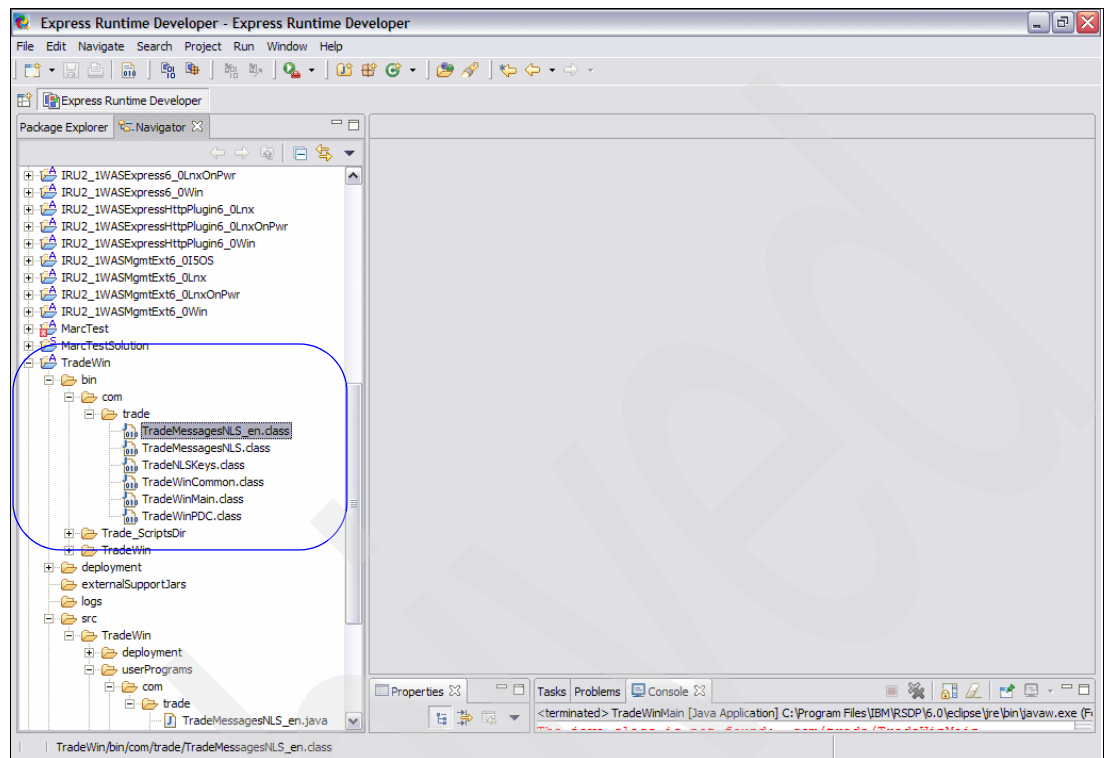


Figure 6-35 Java class files default location

The source codes for all the Java programs used in this solution are included in Appendix A, “Source code for Trade6 user programs and script files on Windows” on page 257.

6.2.4 Completing the Trade6 application project

Now that we have created the user programs and Trade.prop file, we can complete our Trade6 application project:

1. Start Express Runtime Developer by selecting **Start → Programs → IBM Express Runtime 2.1 → Express Runtime Developer**.
2. Copy the Trade.ear and add the script files used by the user programs.
 - a. From Package Explorer, double-click the **TradeWin** folder to expand it.
 - b. Right-click the folder **src/TradeWin/userPrograms** and select **New → Folder** (see Figure 6-36).

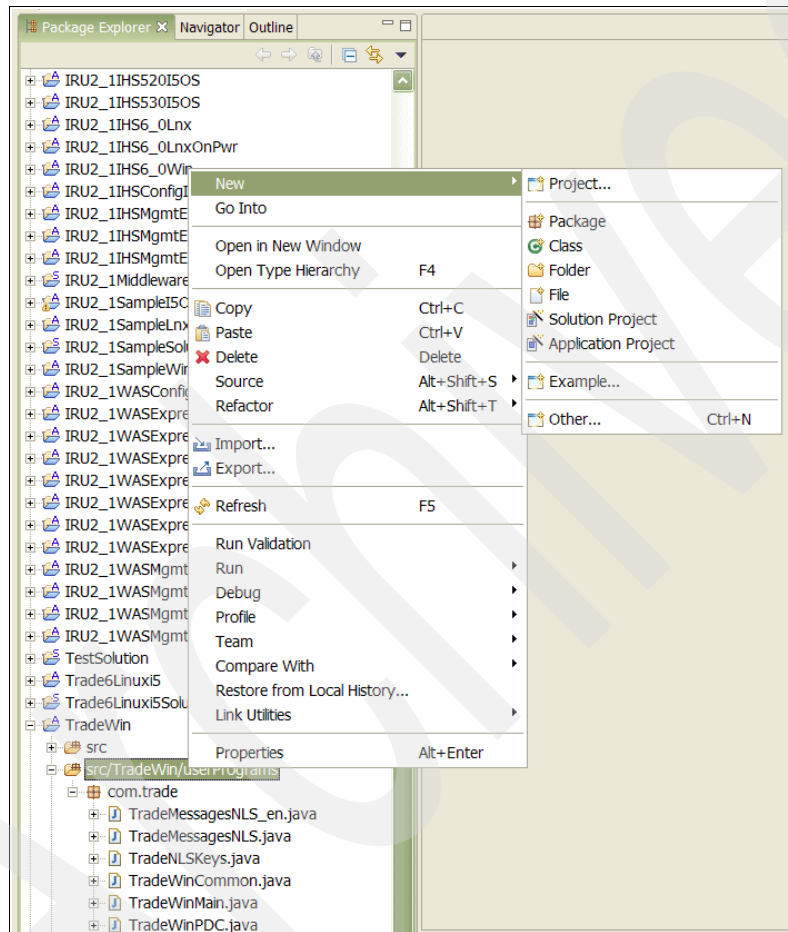


Figure 6-36 Creating a package for Trade6 scripts

- c. In the New Folder window, type Trade_ScriptsDir for the Folder name field. Click **Finish** (Figure 6-37).

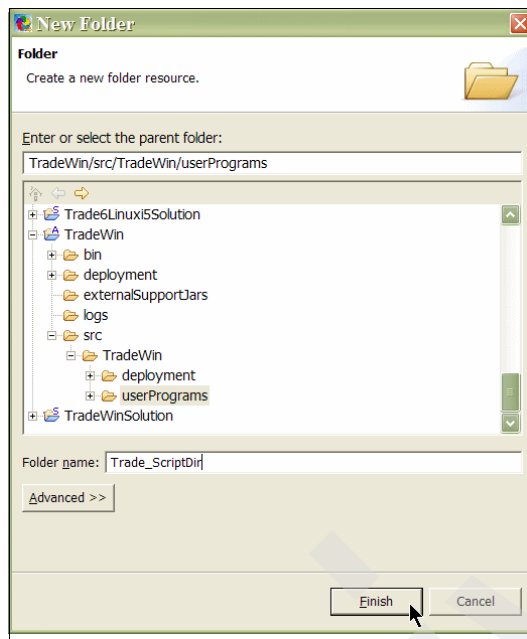


Figure 6-37 New folder for Trade_ScriptsDir

- d. Using Windows Explorer, copy the script files (see Appendix D, “Additional material” on page 435). Paste them in the newly created folder (Trade_ScriptsDir) in Package Explorer. From Package Explorer, the Trade_ScriptsDir content should look like Figure 6-38. The Trade.ear file is located in C:\Trade directory.

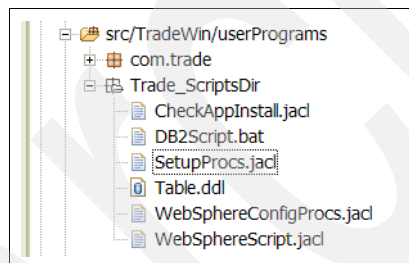


Figure 6-38 Trade_ScriptsDir files

3. Open the TradeWin application project.
 - a. Using Package Explorer, double-click the **TradeWin** folder to expand it.
 - b. Click the **src** folder to expand it.
 - c. Click the **TradeWin** folder.
 - d. Double-click **application.axml** file.

4. Click the **Files** tab. The Application Files Information page appears as shown in Figure 6-39. This tab allows you to define all the files needed for installing your application.

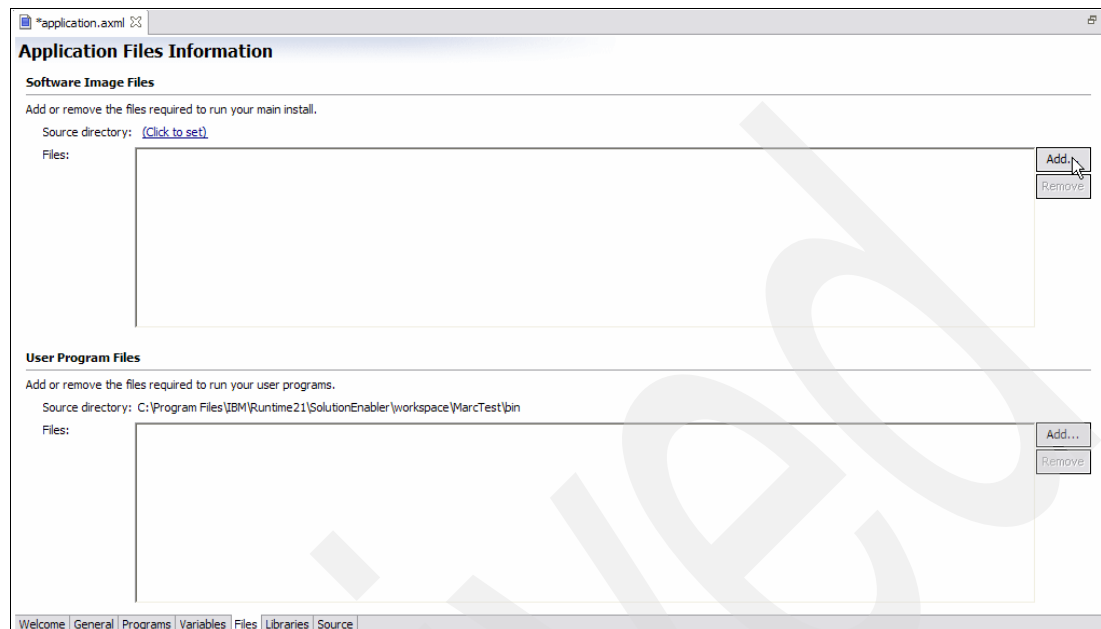


Figure 6-39 Application Files Information tab

5. In the Software Image Files section, click **(Click to set)** to set your source directory where you have the image files.
6. The Browse for Folder window appears (Figure 6-40). Navigate to the folder **C:\Trade**, select it, and click **OK**.

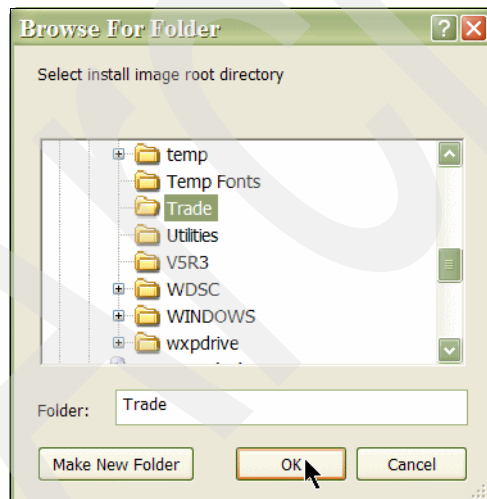


Figure 6-40 Browse for Trade_ScriptsDir folder

Note: At the time of writing this book, there was a known problem. If you have built the deployment packages once and then change the Source directory, you need to:

1. Save your application.
2. Exit the workbench.
3. Restart it for the change to be active.

There is a fix going into the fix pack to address this problem.

4. On the Software Image Files section, click **Add**.
5. In the Add Files window, select **trade.ear**. Click **Finish**.
6. Add compiled user programs:
 - a. In the User Program Files section of the Application Files Information page, click **Add**.
 - b. In the Add Files window, double-click **com**.
 - c. Double-click **trade**. Select all compiled user programs and click **Finish** (Figure 6-41).

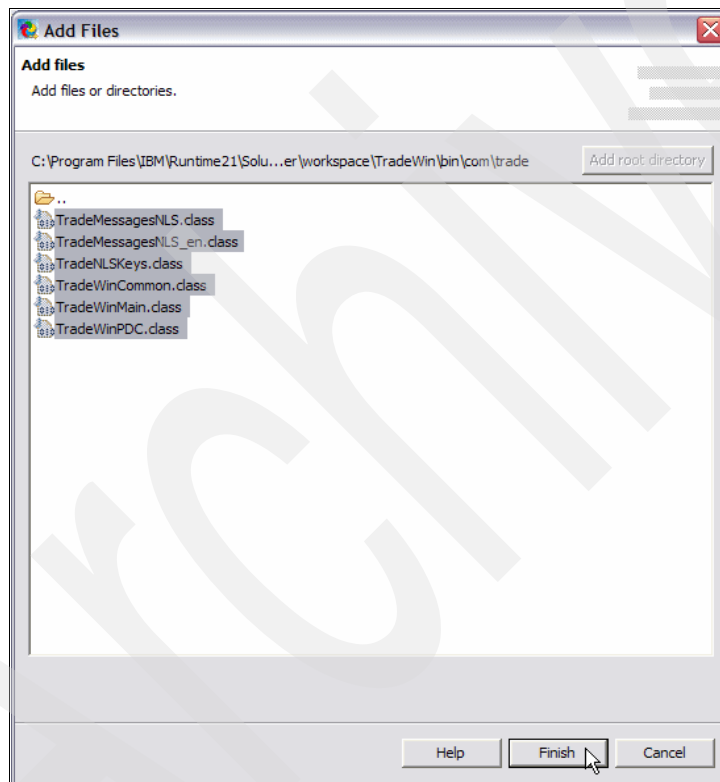


Figure 6-41 Add compiled user programs

7. Add script files

- a. On the User Program Files section of the Application Files Information page, click **Add**.
- b. In the Add Files window, double-click **Trade_ScriptsDir**.
- c. Select all scripts and click **Finish** (Figure 6-42).

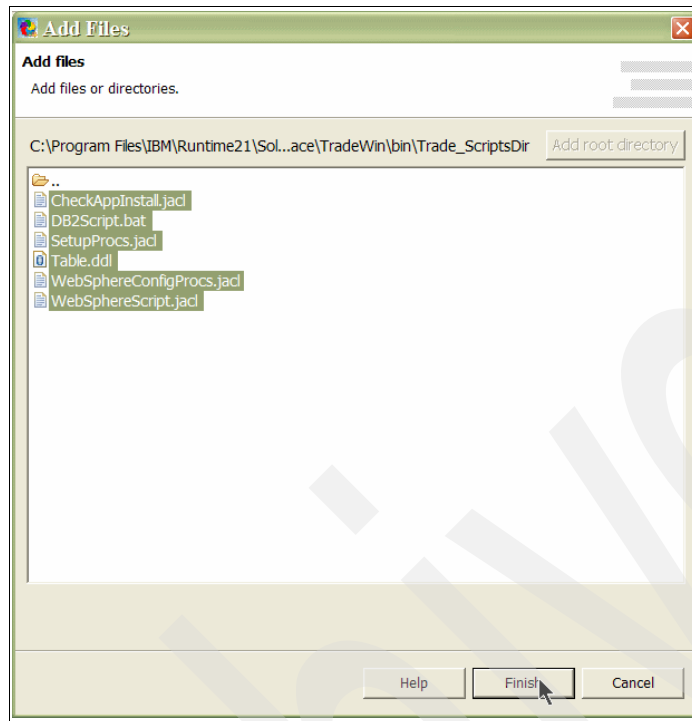


Figure 6-42 Adding scripts

8. The Application Files Information page now looks like Figure 6-43. Save the application project by pressing Ctrl+S.

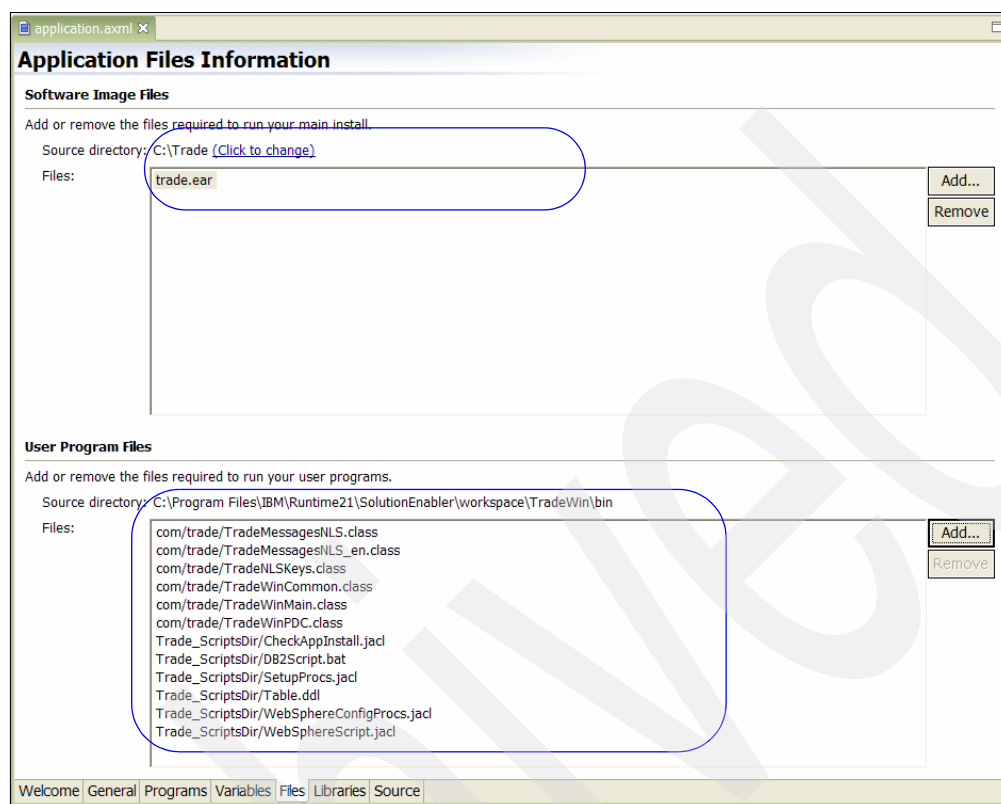


Figure 6-43 Application Files Information page - Complete

6.2.5 Creating Trade6 solution project

The Solution Wrapper Editor gives us a way to create and edit solution wrappers without having to hand-code the raw XML source. Different tabs are provided to define the needed configuration for your solution.

To begin creating the Trade6 solution project, follow these steps now:

1. Open Express Runtime Developer by selecting **Start** → **Programs** → **IBM Express Runtime 2.1** → **Express Runtime Developer**.
2. Create a new Express Runtime application project.

In the Express Runtime Developer window, select **File** → **New** → **Solution Project** (see Figure 6-44).

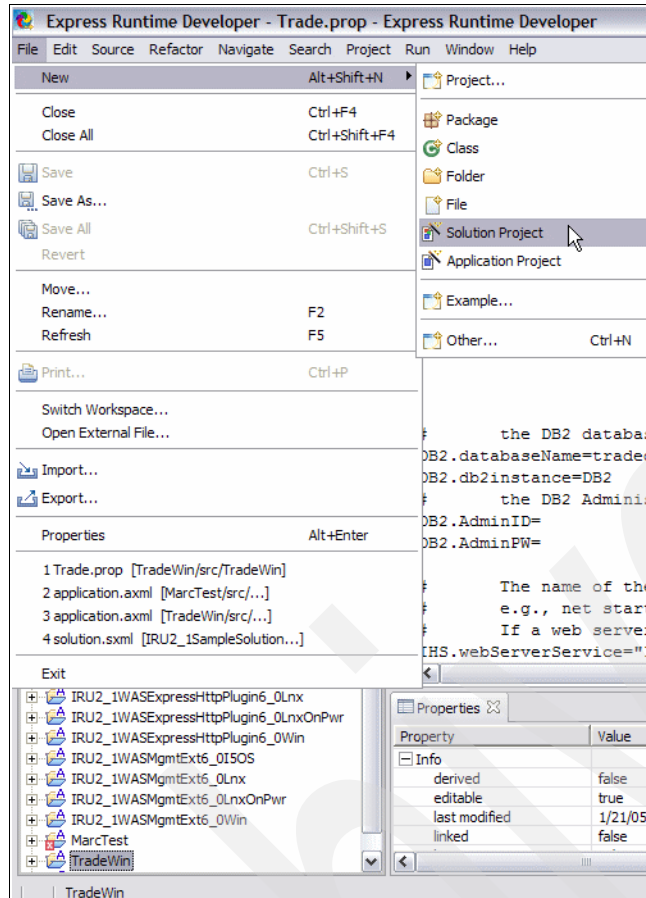


Figure 6-44 Creating a new solution project

3. Type in TradeWinSolution as the Project Name and click **Next** as indicated in Figure 6-45.

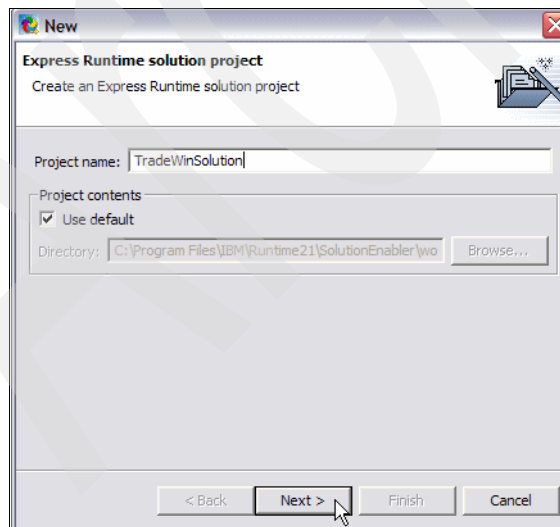
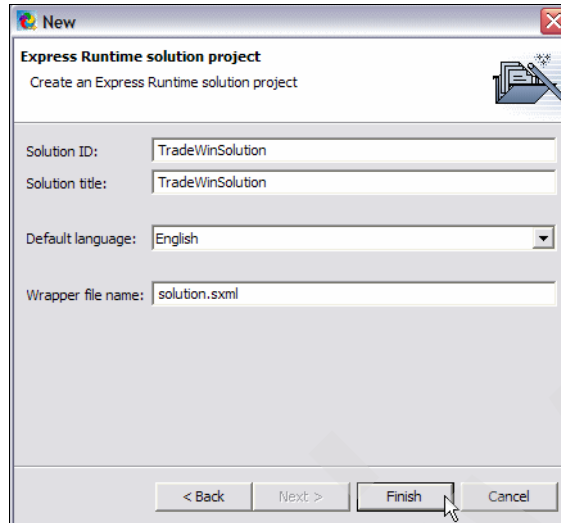


Figure 6-45 New solution project name

4. In the next window, use the following values and click **Finish** as shown in Figure 6-46.

- Application ID: TradeWinSolution
- Solution Title: TradeWinSolution
- Default Language: English
- Wrapper file name: solution.xml



The screenshot shows a Windows-style dialog box titled "New" with a close button in the top right corner. Below the title bar, the text "Express Runtime solution project" is displayed, followed by the instruction "Create an Express Runtime solution project". The dialog contains four input fields: "Solution ID:" with the text "TradeWinSolution", "Solution title:" with the text "TradeWinSolution", "Default language:" with a dropdown menu showing "English", and "Wrapper file name:" with the text "solution.xml". At the bottom of the dialog, there are four buttons: "< Back", "Next >", "Finish", and "Cancel". A mouse cursor is pointing at the "Finish" button.

Figure 6-46 New solution project ID

5. The Express Runtime Developer window now shows you the Solution Wrapper Editor Welcome page in the right pane. In Package Explorer, a newly created folder named TradeWinSolution is shown. Figure 6-47 reflects the newly created solution project. Specify the needed components to develop the solution wrapper using the tabs on the Welcome page.

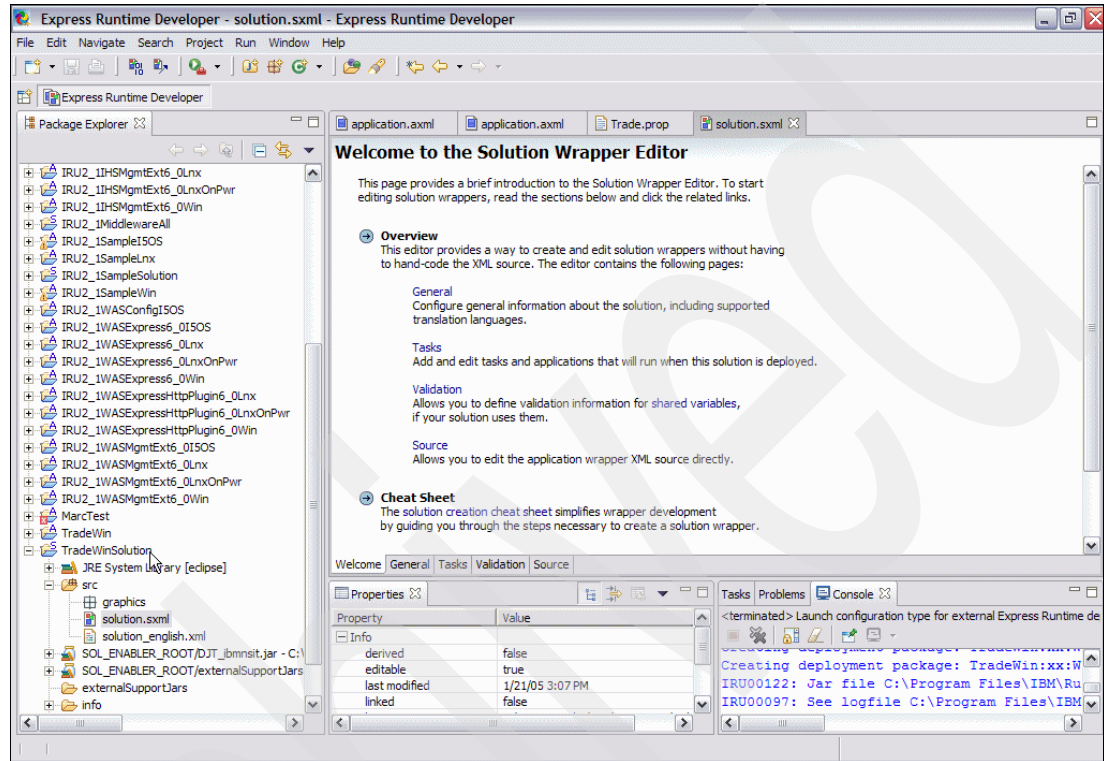


Figure 6-47 Welcome page for Solution Wrapper Editor

6. Click the **General** tab. The window in Figure 6-48 appears. It shows you the fields where you can specify the general information of your solution project. Set the values as follows:
 - *ID: TradeWinSolution
 - *Title: TradeWinSolution
 - Version: 6.0
 - About Screen Text: `<![CDATA[<html><p align="left"><p style="margin-left:25px">
Trade Solution
</style></align></html>]]>`
 - Welcome Screen Title: Trade V6 Solution
 - Welcome Screen Text: The solution to deploy all middleware, the Trade application, and Console for Express Runtime.
 - Task group selection prompt: Select which tasks you would like to deploy
 - Default Language: English

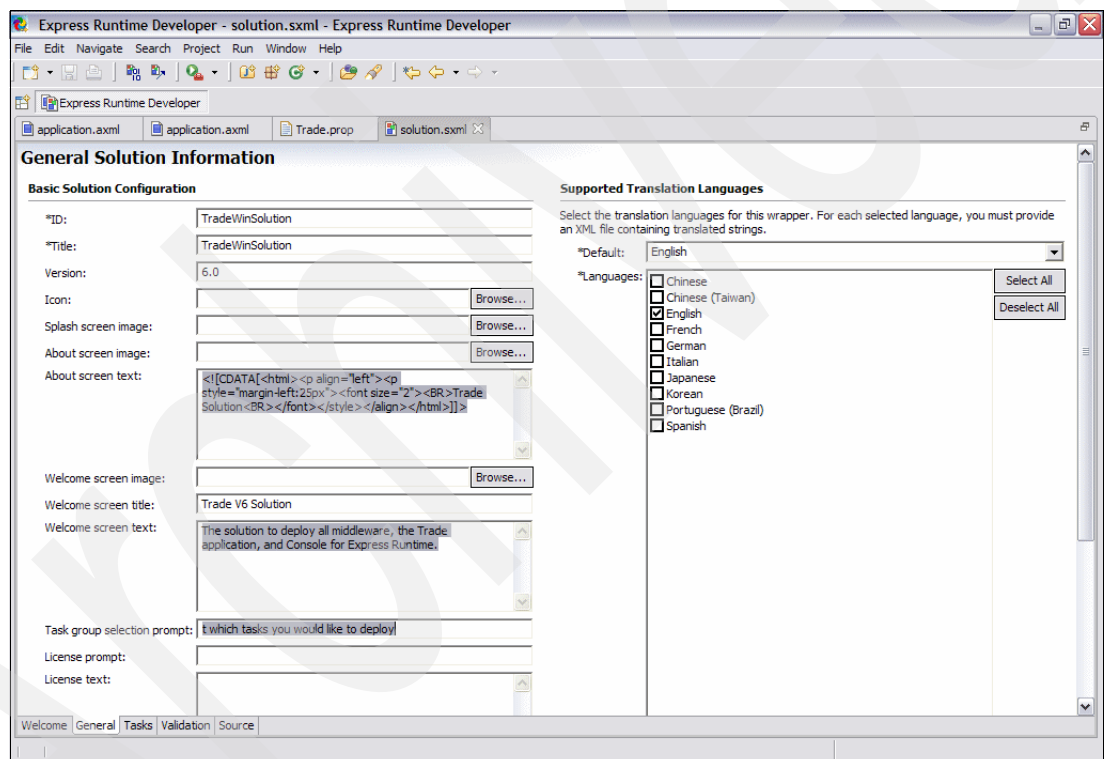


Figure 6-48 General Solution Information

- Click the **Tasks** tab. This tab, shown in Figure 6-49, lets you define the tasks needed to install the solution (such as installation of middleware, and other application projects). Click **Add**.

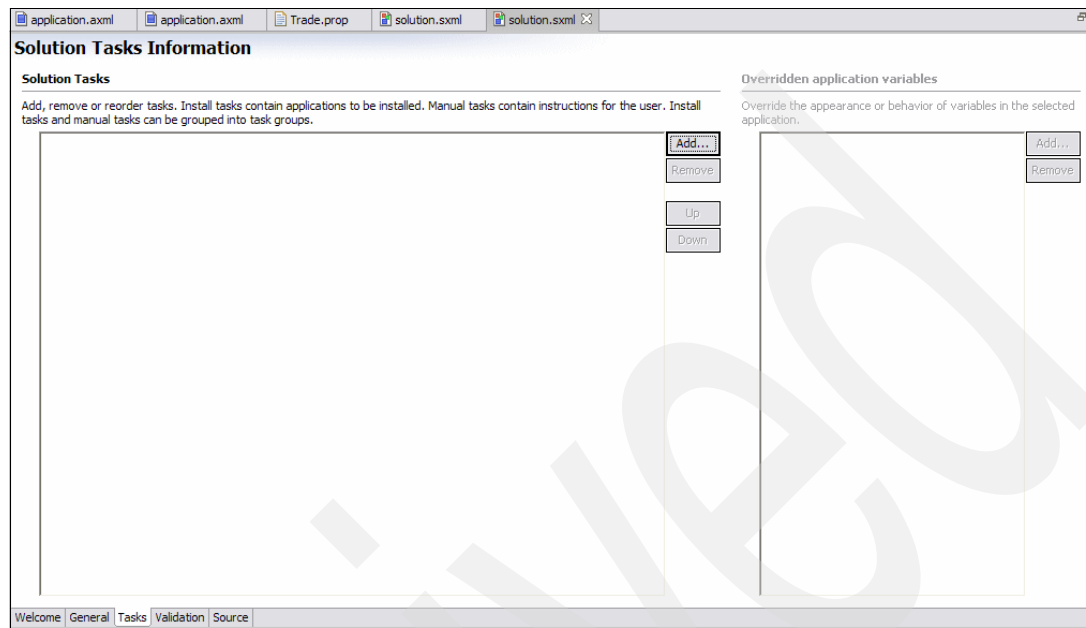


Figure 6-49 Solution Tasks information

- The Add window appears (see Figure 6-50). Select **Task Group** and click **Next**.

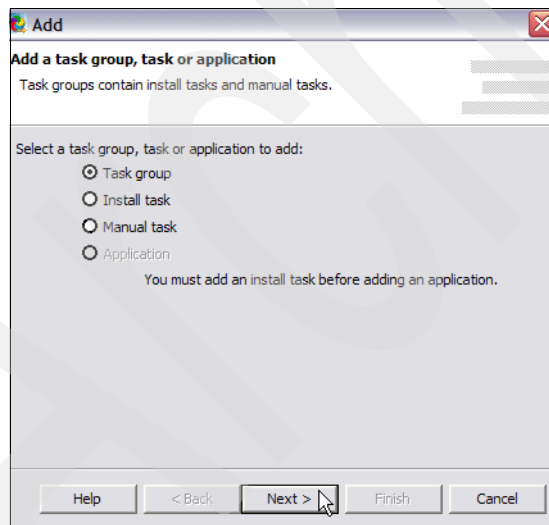


Figure 6-50 Add Task pop-up window

9. Provide the following values as shown in Figure 6-51. Click **Finish**.
- Task group title: Trade Application
 - Task group prompt: Select the platforms where the middleware and Trade6 application will be installed.

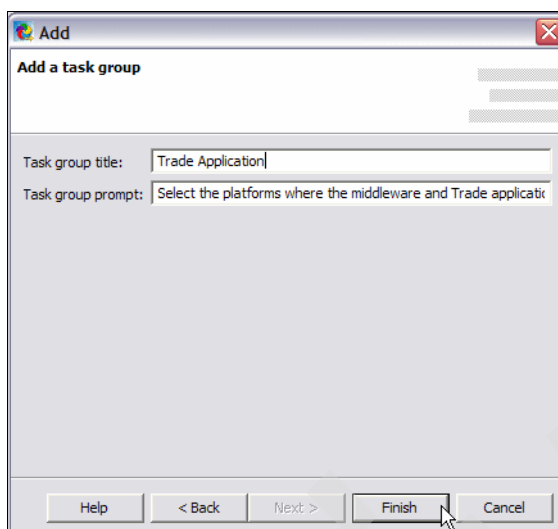


Figure 6-51 Adding the Trade task group

10. To configure the install tasks, right-click the **Task Group** and select **Add Install Task**.
11. Set the following values as shown in the Add an install task panel.
- Parent task group: **Trade Application**
 - Task description: Trade Application and Middleware for Windows
 - Operating System: **Windows**
 - Enable the **Launch the “Add Application” wizard after adding this task** option

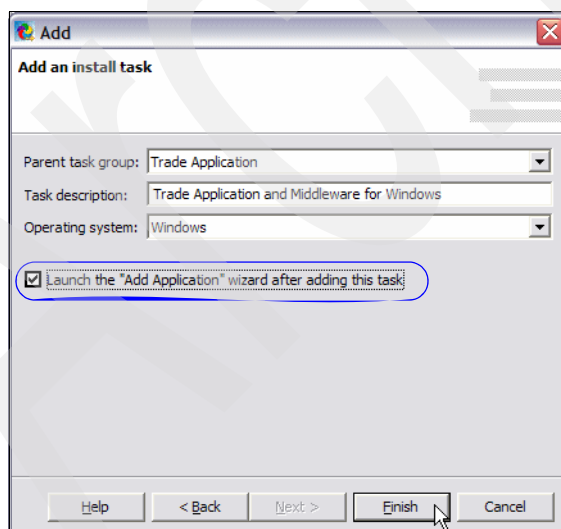


Figure 6-52 Adding Install Task

Note: Since we are deploying Trade6 on a Windows platform, we selected Windows in the Operating system field from this window. If you need to deploy an application to a different platform (such as Linux, Linux on POWER, or OS/400), you can select a different operating system.

Click **Finish**.

12. In the Add Applications window, select all applications needed by Trade6 as shown in Figure 6-53. Click **Finish**.

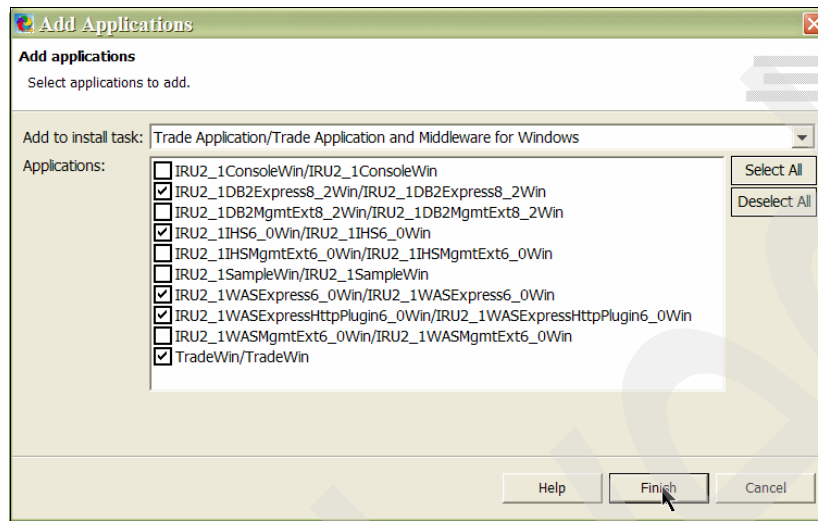


Figure 6-53 Adding applications to the installation task

Note: Having all components defined in a single install task allows for deployment on one target system. If you need to divide the components to multiple system environments, you can create separate install tasks and assign each task to the corresponding target system during deployment. You might also need to alter your user programs, scripts, or both to ensure that they can perform additional configuration that may be required as a result of deploying across multiple systems.

A Task Group is a way to group related tasks and assist in selecting which tasks need to be deployed. Optionally you can create a project without a task group that only has install tasks in a solution. For instance, the MiddlewareAll solution has a "Database" Task Group which contains three tasks. The three tasks are for DB2 installation on each platform (Linux, Linux on POWER, and Windows). During deployment, the first configuration panel asks what the user wants to deploy. If the deployer chooses "Database" (the task group), the next panel displays the three tasks where the user can select which platform on which they want to deploy DB2.

Your Solution Tasks Information page should now look like Figure 6-54.

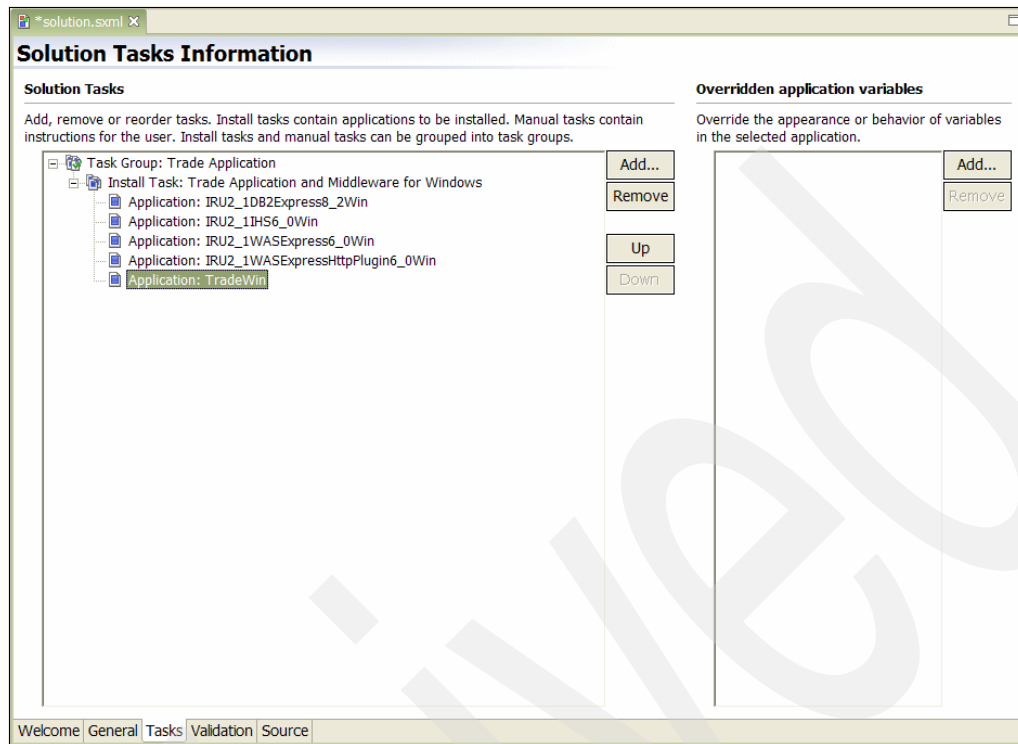


Figure 6-54 Solution Tasks Information panel with entries

Note: If you plan to use the Console for Express Runtime as part of your solution, you need to add the management extensions for each middleware component and the Console for Express Runtime application projects (see Figure 6-55).

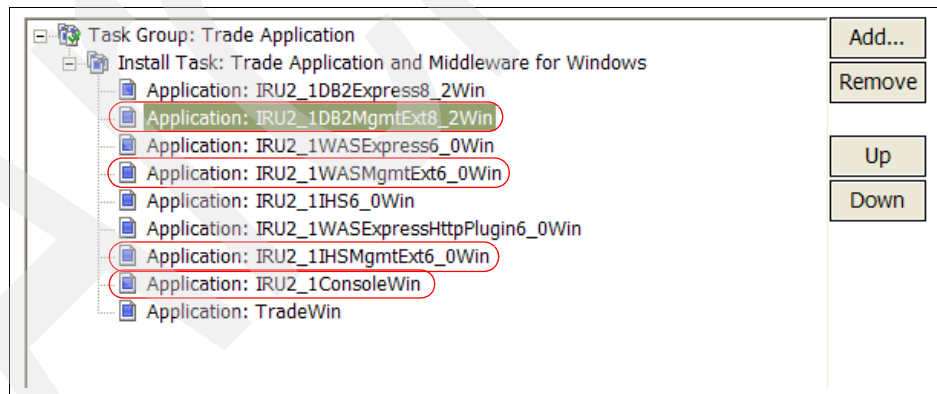


Figure 6-55 Solution with the Console for Express Runtime

13. In your application, you may need to reorder tasks. You can select a task and click the **Up** and **Down** buttons to organize the task entries. This is frequently required to ensure that the deployment works correctly. For example, it is not possible to deploy the trade.ear application or create the Trade database table before first deploying WebSphere Application Server Express and DB2 Express.

14. From the Install Task list, select **Application: IRU2_1DB2Express8_2Win**.

Note: For the purpose of showing an example on how to configure variable overrides, we choose to work on the DB2Express application only. Variable overrides are used here to share the value that the deployer enters between application projects. This avoids asking the Deployer to provide the DB2 userid and password multiple times.

The Trade6WinSolution.zip file (see Appendix D, “Additional material” on page 435) has the full configuration of the Trade6 application and solution projects. You may refer to this file to get the configuration of each variables needed to deploy Trade6.

15. On the Overridden application variables section, click **Add**.

16. Select **username** and **password** from the list and click **Next** (see Figure 6-56).

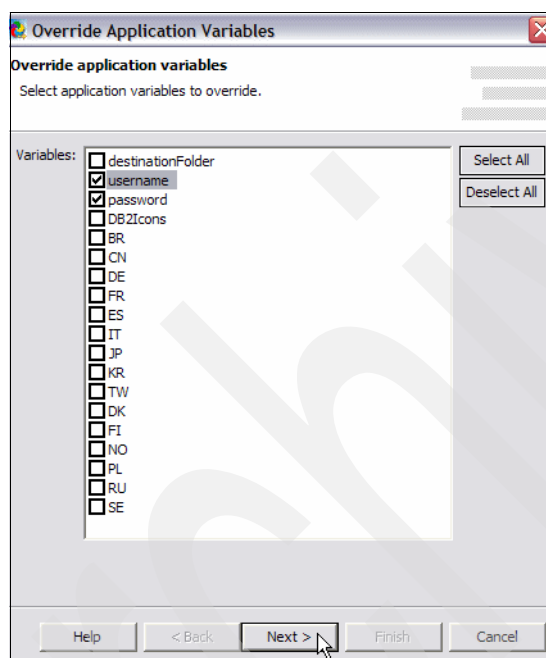


Figure 6-56 Override Application Variables

17. On the Override Applications Variable: Override username window (Figure 6-57), select the following values, and click **Next**.
 - Behavior: **Share the value of username with other variables**
 - Shared as: **DB2AdminUsernameWin**

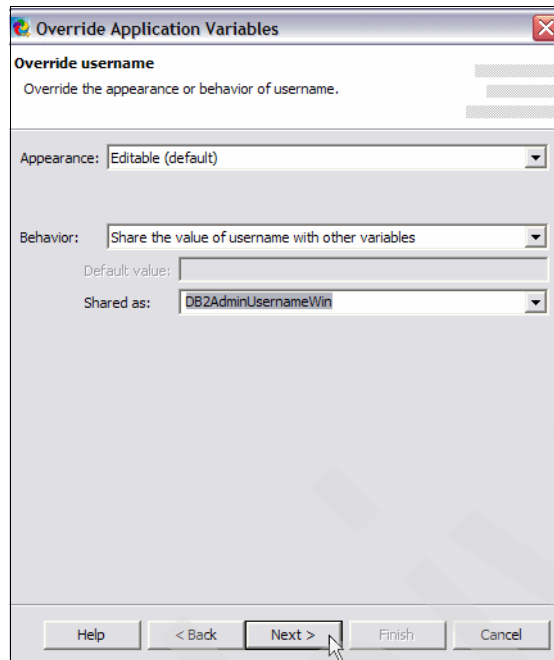


Figure 6-57 Override variable

18. On the Override Applications Variable: Override password window, select the following values, and click **Finish**.
 - Behavior: **Share the value of password with other variables**
 - Shared as: **DB2AdminPasswordWin**
19. Click the **Validation** tab. This shows the Shared Variables Validation Information page.
20. In the Shared Variables section, click **Add**.
21. In the Add Shared Variables window, click **Select All**. Click **Finish**.

Note: As an example on how to configure shared variables, we configure the variable DB2AdminUsernameWin only.

22. Select DB2AdminUsernameWin from the Shared variables list and set the following values:
 - Default value: db2admin
 - Minimum length: 1
 - Maximum length: 30
 - Validation rules:
 - Valid characters: "@#\$_abcdefghijklmnopqrstuvwxy0123456789"
 - Invalid values: "ADMINS"
 - Invalid values: "GUESTS"
 - Invalid values: "USERS"
 - Invalid values: "PUBLIC"
 - Invalid values: "LOCAL"
 - Invalid prefix: "IBM"
 - Invalid prefix: "SQL"

Invalid prefix: "SYS"
Invalid prefix: "_"

The Validation page should look like Figure 6-58.

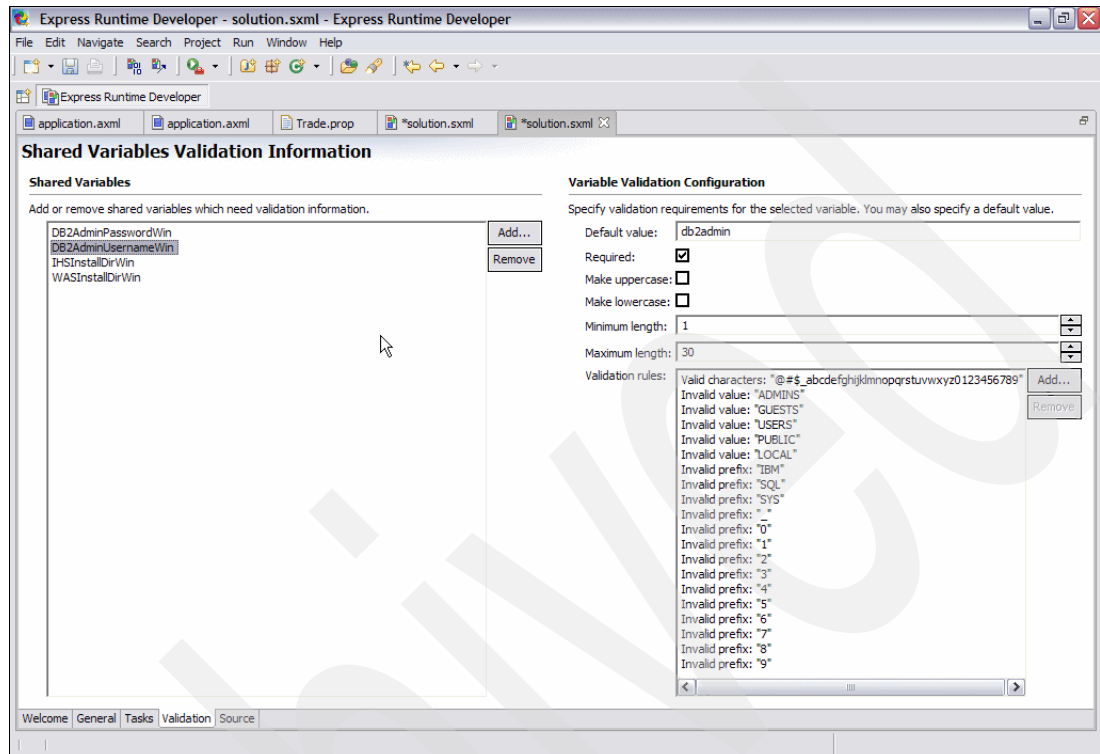


Figure 6-58 Validation rules

23. Save your solution project by selecting **File** → **Save** or by pressing **Ctrl+S**.

6.2.6 Building the Trade6 solution

With all the XML configurations and user programs defined, the next task is to use Express Runtime Developer to build the projects and make it ready for deployment.

During generation, the solution project makes a reference to the application projects in its definition. And for this reason, each application project in the solution must be built first before building the solution project.

For our Trade6 example, the following middleware application projects are needed:

- ▶ IRU2_1DB2Express8_2Win
- ▶ IRU2_1WASExpress6_0Win
- ▶ IRU2_1IHS6_0Win
- ▶ IRU2_1WASExpressHttpPlugin6_0Win

We do not need to generate these projects, as they are already provided in Express Runtime V2.1.

Once the application and solution projects are generated, the solution needs to be exported as the last piece of enabling the solution for deployment. This is covered in Chapter 7, "Packaging a solution" on page 165.

Building the Trade6 application project

The following tasks guide you in building the Trade6 application project:

1. From Package Explorer, right-click the **TradeWin** folder and select **Generate Application** (see Figure 6-59).

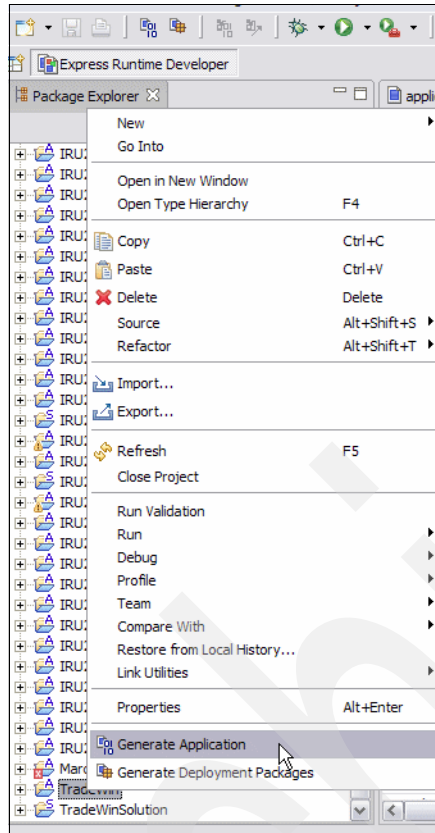


Figure 6-59 Option for Generating the application

Note: The console window at the lower right area of Express Runtime Developer shows some information during the process of generating the application project.

2. The Application Generation Successful window appears stating that the application has been generated successfully (see Figure 6-60). Click **OK**.

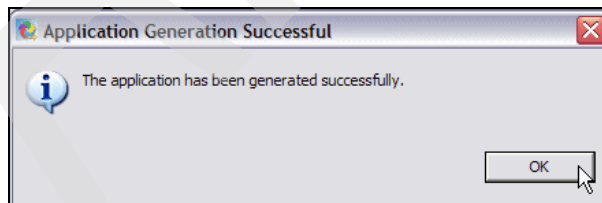


Figure 6-60 Pop-up window for successful generation of application

The binary file TradeWin_win.ser is created in the <ER workspace dir>\TradeWin\bin\TradeWin folder (see Figure 6-61). This is the result of generating the deployment package for TradeWin using application.xml.

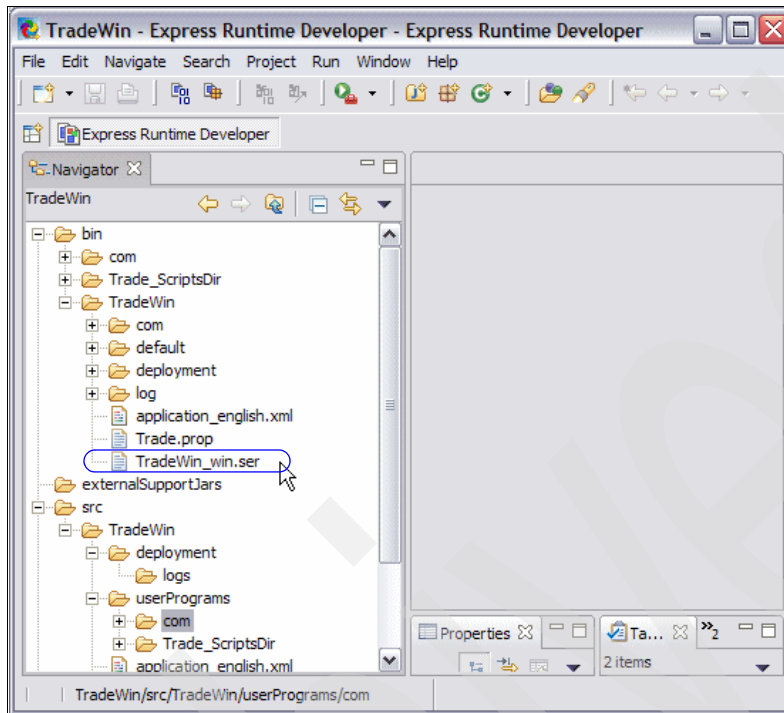


Figure 6-61 Location of TradeWin_win.ser file

3. Produce the Deployment Package.
 - a. From Package Explorer, right-click the **TradeWin** folder and select **Generate Deployment Packages**.

Note: The console window at the lower right area of Express Runtime Developer shows some information during the process of generating the deployment package.

- b. A pop-up window appears (Figure 6-62) stating that the package has been generated successfully. Click **OK**.

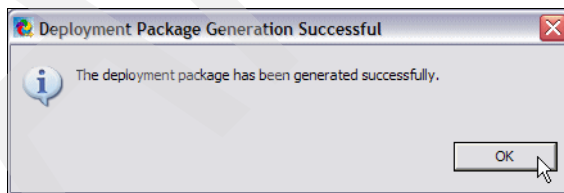


Figure 6-62 A pop-up message for successful generation of a development package

This creates the jar file tradewin.xx.jar in the folder <ER workspace dir>\IRU_common_resources\mediaJars. The application EAR/WAR file is packaged into this JAR file.

Building the Trade6 solution project

Once the application project has been generated, follow these steps in generating the solution project:

1. From Package Explorer, right-click the **TradeWinSolution** folder and select **Generate Solution** (see Figure 6-63).

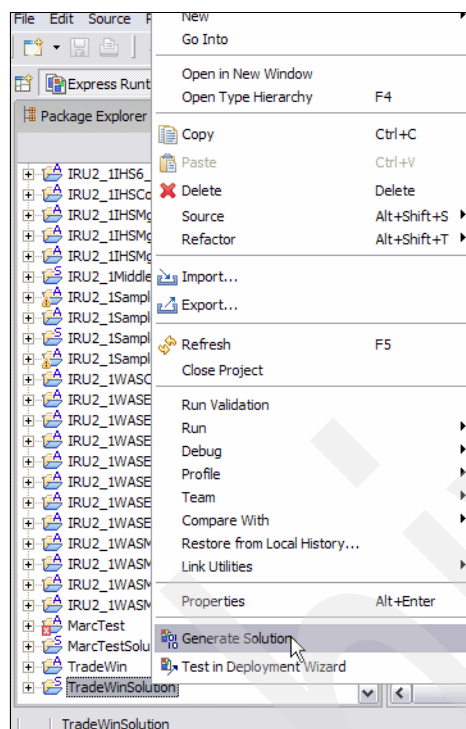


Figure 6-63 Option for Generating Solutions

2. You see a series of pop-up windows while the solution is being created. The final window shows you that the solution has been generated (see Figure 6-64). Click **OK**.

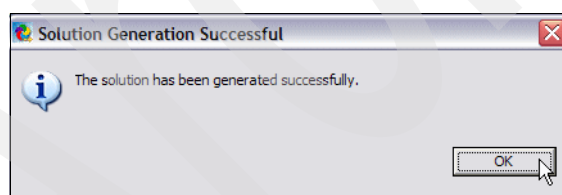


Figure 6-64 Solution Generation Successful

The binary file TradeWinSolution.ser is created in <ER workspace dir>\TradeWinSolution\bin (see Figure 6-65). This is the result of generating the solution from the TradeWinSolution wrapper solution.xml.

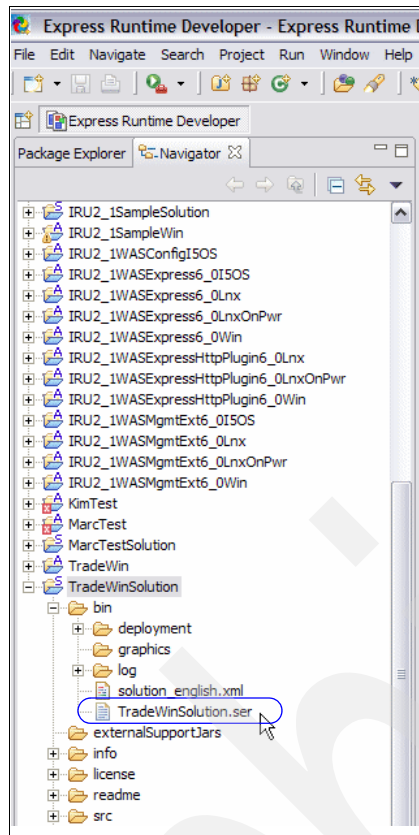


Figure 6-65 TradeWinSolution.ser location

3. The solution is now ready. You can use the Deployment Wizard to test your solution. From Package Explorer, right-click **TradeWinSolution** and select **Test in Deployment Wizard** (see Figure 6-66).

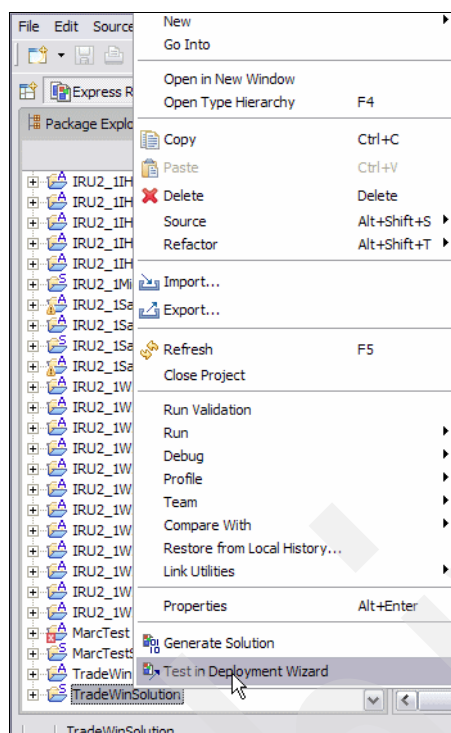


Figure 6-66 Testing a solution

6.3 Developing Wrappers for Trade6 for Linux on POWER

This section demonstrates the process involved in developing an application and carrying out deployment of the same application on Power Linux. We started the development from reviewing the IRU2_1SampleLnx Application Project. Nearly 80% of the code in this project can be reused for developing the Trade6 wrapper for Linux on POWER.

First, we created an identical project structure. Second, we copied most of the code (environment, methods, and properties) from the sample project and replaced references to Sample Linux classes with our own class names. Third, we created a folder for the JACL and DB2 Scripts and implemented them into our application project.

Essentially, the development process is identical to the one described in 6.2, “Developing the Trade6 wrapper for Windows” on page 93.

Next we describe the contents of the application project, and provide a brief description of the files that we used for our solution:

- **TradeLnxPDC** — Predeployment checker

Most of the code in this file has been reused from the IRU2_1SampleLnx code and is used to determine whether middleware components are already installed. Sample File is provided in Example: B-4, “TradeLnxPDC.java” on page 344.

- **TradeLnxMain** — Main Program

Some of the code in this file has been reused from the IRU2_1SampleLnx code and is the Main Program which is used to call other script files. However, it also contains additional

code which is application specific program as illustrated in Example 6-11. The full source code is provided in Example B-3 on page 326.

Example 6-11 Snippet of code in TradeLnxMain Program

```
String [] command1 = {"/bin/su","-", "db2inst","-c","/irul/Trade_ScriptsDir/DB2Script.sh  
tradedb /irul/Trade_ScriptsDir/ db2inst db2inst > /home/db2inst/DEBUG_OUTPUT"};
```

► **TradeLnxCommon — Common Program**

Most of the code in this file has been reused from the IRU2_1SampleLnx code. The functions included here are used by PDC and Main Programs for carrying out various functions. The full source code is provided in Example: B-5, “TradeLnxCommon.java” on page 353.

► **Trade.prop**

This is an application specific file for providing a response to the Trade6 application. Example 6-12 is a snippet of the file.

Example 6-12 Snippet of code in Trade.prop Response File

```
TRADE.WASscript=WebSphereScript.jacl  
  
# The name of the DB2 script to be run  
TRADE.DB2script=DB2Script.sh  
  
# the install EAR or WAR file  
WAS.appInstallFile=trade.ear  
  
# the DB2 database name and instance  
DB2.databaseName=tradedb  
DB2.db2instance=DB2
```

► **DB2Script.sh and Table.ddl**

These are the application specific files for creating the database. The full source code is provided in Example B-9 on page 386.

► **CheckAppInstall.sh**

This is an application specific code for checking whether the application already exists on the system. The full source code is provided in Example B-6 on page 357.

► **SetupProcs.jacl**

Most of the code in this file has been reused from the IRU2_1SampleLnx code to set up variables for response file. The full source code is provided in Example B-11 on page 389.

► **TradeMessagesNLS.java, TradeNLSKeys.java and TradeMessagesNLS_en.java**

Most of the code in this file has been reused from the IRU2_1SampleLnx code and has been used for exception handling in the deployment. The full source code is provided in Appendix B, “Source code for Trade6 user programs and script files for Linux on POWER” on page 323.

► **WebSphereScript.jacl and WebSphereConfigProcs.jacl**

Most of the code in this file has been reused from the IRU2_1SampleLnx code and is a sample code for creating JDBC connection to the database. The full source code is provided in Appendix B, “Source code for Trade6 user programs and script files for Linux on POWER” on page 323.

A detailed explanation for all the above files can be found in 6.2.3, “Creating Trade6 user programs” on page 113.

6.3.1 Preparing for Trade6 deployment

Refer to Appendix D, “Additional material” on page 435 for instructions on downloading a zip file. This file contains the Trade6 application in EAR format plus all the files and projects we need for demonstration. You can use this example to pattern your own solution with IBM Express Runtime V2.1.

Trade6 Deployment has been tested under the following environment:

- ▶ Production Platform — SUSE Enterprise Linux 9 — 2.6.5-7.135-pseries64
- ▶ Express Runtime version — Gold Media CD 2.1
- ▶ i5 system with 6GB RAM, 0.4 CPU (minimum)
- ▶ Development Platform — Windows

6.3.2 Code customization before deployment

Listed here are the changes that may need to be done depending on the deployment environment:

- ▶ WebSphereScript.jacl

The DB2 DAS instance port and IP Address may need to be changed depending on the environment:

```
createDB2orCloudscapeDatasource $datasourceName "jdbc/${datasourceName}" 60 $jdbcProvId
db2 $authAliasName "Trade6 Datasource" $nodeScope $databaseName [list 4 localhost 50001]
```

The db2UserID and db2Password may be different for JAAS Authorization:

```
createJAASDataAuth $authAliasName $db2UserId $db2Password $authAliasDesc
```

- ▶ SetupProcs.jacl

The DB2 User and Password used here is the same for both JAAS and JMS Connectivity and may need to be customized as per the environment of deployment:

```
set DefaultOSUser [string trim [$props getProperty DB2.UserID ]]
set DefaultOSPasswd [string trim [$props getProperty DB2.UserPW ]]
set databaseName [string trim [$props getProperty DB2.databaseName ]]
set db2instance [string trim [$props getProperty DB2.db2instance ]]
puts " db2Password = ***** "
```

6.3.3 Deploying the Trade6 Solution

Given here are the basic steps for deployment the Trade6 application:

- ▶ Install the “unzip” rpm on the i5 system.
- ▶ Create Trade6 application project as demonstrated in 6.2.1, “Creating the Trade6 application project” on page 93.
- ▶ Create Trade6 user programs as demonstrated in 6.2.3, “Creating Trade6 user programs” on page 113.
- ▶ Create Trade6 Solution as demonstrated in 6.2.5, “Creating Trade6 solution project” on page 132.
- ▶ Deploy the Trade6 Solution from the development system.

Note: The Trade6 solution included in Appendix D, “Additional material” on page 435, does not contain the middleware deployment components. The custom solution that you build would need to include the middlewares as illustrated in 6.2.5, “Creating Trade6 solution project” on page 132.

6.3.4 Files required for Trade6 application

In case you intend to copy the files manually (instead of importing the Solution), Table 6-1 gives file locations where the files should be copied.

Note: In the following table, <ER Workspace Dir> = C:\Program Files\IBM\Runtime21\Solution Enabler\workspace.

Table 6-1 List of files to be manually copied before starting deployment

Filename	Path where the files should be kept
trade.prop	<ER Workspace Dir>\Trade6linuxi5\src\Trade6linuxi5\
application.xml	<ER Workspace Dir>\Trade6linuxi5\src\Trade6linuxi5\
CheckApplInstall	<ER Workspace Dir> \Trade6linuxi5\src\Trade6linuxi5\userPrograms\Trade_ScriptsDir\
DB2Script.sh	<ER Workspace Dir> \Trade6linuxi5\src\Trade6linuxi5\userPrograms\Trade_ScriptsDir\
IRU_EditMap.jacl	<ER Workspace Dir> \Trade6linuxi5\src\Trade6linuxi5\userPrograms\Trade_ScriptsDir\
SetupProcs.jacl	<ER Workspace Dir> \Trade6linuxi5\src\Trade6linuxi5\userPrograms\Trade_ScriptsDir\
trade.ear	<ER Workspace Dir> \Trade6linuxi5\src\Trade6linuxi5\userPrograms\Trade_ScriptsDir\
Trade.ddl	<ER Workspace Dir> \Trade6linuxi5\src\Trade6linuxi5\userPrograms\Trade_ScriptsDir\
WebConfigProcs.jacl	<ER Workspace Dir> \Trade6linuxi5\src\Trade6linuxi5\userPrograms\Trade_ScriptsDir\
WebSphereScripts.jacl	<ER Workspace Dir> \Trade6linuxi5\src\Trade6linuxi5\userPrograms\Trade_ScriptsDir\
TradeLnxDPC.java	<ER Workspace Dir> \Trade6linuxi5\src\Trade6linuxi5\src\Trade6linuxi5\userPrograms\com\trade\
TradeLnxDMain.java	<ER Workspace Dir> \Trade6linuxi5\src\Trade6linuxi5\src\Trade6linuxi5\userPrograms\com\trade\
TradeLnxDCommon.java	<ER Workspace Dir> \Trade6linuxi5\src\Trade6linuxi5\src\Trade6linuxi5\userPrograms\com\trade\
TradeMessagesNLS.java	<ER Workspace Dir> \Trade6linuxi5\src\Trade6linuxi5\src\Trade6linuxi5\userPrograms\com\trade\
TradeMessagesNLS_en.java	<ER Workspace Dir> \Trade6linuxi5\src\Trade6linuxi5\src\Trade6linuxi5\userPrograms\com\trade\
TradeNLSKeys.java	<ER Workspace Dir> \Trade6linuxi5\src\Trade6linuxi5\src\Trade6linuxi5\userPrograms\com\trade\
TradeMessagesNLS.java	<ER Workspace Dir> \Trade6linuxi5\src\Trade6linuxi5\src\Trade6linuxi5\userPrograms\com\trade\

6.4 Developing a wrapper for the WebFacing application

In this section we provide a brief overview of the development process. The process itself is very similar to the one described in 6.2, “Developing the Trade6 wrapper for Windows” on page 93. We only point to several unique steps in developing this application project.

6.4.1 Before developing a wrapper

We used the Flight400 RPG application for this example. For more information about this WebFacing application, see Chapters 4 and 5 in *Mastering the IBM WebFacing Tool*, SG24-6331. Appendix D, “Additional material” on page 435, contains information for downloading and installing the Flight400 sample application on your system. You have to perform this step before trying any other steps in this example.

The next step in developing a wrapper for this application is to build a wsadmin script to install and configure the WebFacing application in IBM WebSphere® Application Server - Express Version 6 on iSeries. You can see the script's source code in “The WebSphereScript.jacl file” on page 402.

We used 2 JAACL files from the *IRU2_1SampleI5OS* project as the example for writing the wsadmin script:

- ▶ IRU_WebSphereConfigProcs.jacl
- ▶ IRU_WebSphereScript.jacl

Next we manually created a WAS V6 profile on our test iSeries and ran our script to install the Flight400 WebFacing application. After fixing several errors, we ran a successful test. From this point on we worked on building a wrapper.

6.4.2 Creating an application project

Creating an application project in Express Runtime Developer is a required step. See more details in 6.2.1, “Creating the Trade6 application project” on page 93. After we defined the initial parameters, we have information in the application.xml file as shown in Figure 6-67 and Figure 6-68.

application.xml x

General Application Information

Basic Application Configuration

*ID: Flight400 [Click to change](#)

*Application name: Flight400

*Version: 1.0

Installation time (minutes): 12

Provider name:

License text:

Supported Translation Languages

Select the translation languages for this wrapper. For each selected language, you must provide an XML file containing translated strings.

*Default: English

*Languages:

- ☒ English
- ☐ French
- ☐ German
- ☐ Italian
- ☐ Japanese
- ☐ Korean
- ☐ Portuguese (Brazil)
- ☐ Spanish
- ☐ Simplified Chinese
- ☐ Traditional Chinese

Select All
Deselect All

Advanced Application Configuration

Deployment package protected: ☐

Deployment package name:

Configuration instructions:

Supported Operating Systems

Select the operating systems supported by this application.

- ☐ Linux
- ☐ Linux on POWER
- ☒ OS/400 (i5/OS)
- ☐ Windows

Select All
Deselect All

Welcome General Programs Variables Files Libraries Source

Figure 6-67 General tab

User Programs Information

User Programs

Add, remove or edit the user programs defined in the application. The main program is required.

User Programs	
Predeployment Checker	Add...
Main Program	Remove
Exit Program	

Basic Program Configuration

*Program type: Java program

*Program: com.ibm.flight400.SamplePDC Set...

Custom Program Options

System command: ☐

Java Program Options

Additional classpath:

Advanced Program Configuration

Timeout (minutes): 5

Wait for completion: ☒

Program reboots: ☐

Force reboot: ☐

Environment variables: Add... Remove

Program Success Type

Specify how to determine if the program ran successfully.

Success type: Check return code

Search strings: Add... Remove

Program Arguments

Response file: Flight400.prop Browse...

Log file name: SamplePDC.log

Arguments: <response file> V5R2 Add... Remove Up Down

Welcome General Programs Variables Files Libraries Source

Figure 6-68 Programs tab

We defined three user programs:

- ▶ SamplePDC.java — predeployment checker
- ▶ SampleMain.java — the Main program that drives installation of the application
- ▶ SampleExit.java — the exit program to start the WebFacing server on iSeries

6.4.3 Developing user programs

The next step is to create the user programs skeletons. We select *com.ibm.flight400* as the package name for our programs. Then we create three Java classes according to our Programs tab selection.

We review the IRU2_1SampleI5OS application project. Many user programs' methods are very similar to what we have to do. So, we selectively copy and paste these methods into our Java programs.

Important: If you use the copy-and-paste technique, make sure you replace the references to the sample project names with your own class and variables names.

To follow the same project structure and to adhere to good programming practice, we create several additional Java classes (see Figure 6-69).

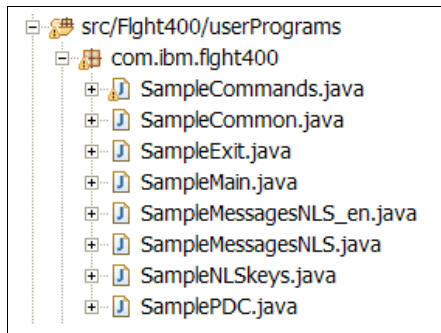


Figure 6-69 Java classes in the wrapper

The content for the messages and NLS keys classes is copied from SampleMessagesNLS.java and SampleNLSKeys.java in the IRU2_1SampleI5OS project.

Properties file

We create the properties file (response file) based on the parameters that are required by our user programs and script. We place this file under the same folder as application.xml (see Figure 6-70).

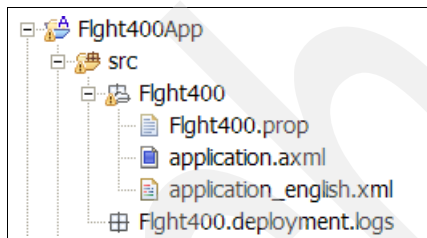


Figure 6-70 The Flight400 properties file

Example 6-13 shows the content of this file.

Example 6-13 The source of the Flight400.prop file

```
# This is the properties for the sample application that comes with
# Redbook.
# *****

#-----
# HINTS:
#   - Use forward slash in directory entries
#
# Note that some of these values are set in Main
#-----

#       The name of the WebSphere configuration script to be run
Flight400.WASscript=WebSphereScript.jacl

#       The scripts are part of the userPrograms fileList in the application.xml.
```

```
# The scriptsDir is the directory under which the scripts were placed, as
# specified in the application.xml.
Flight400.scriptsDir=Flight400_Scripts

# Variables used for WAS
WAS.version=6.0.0.0
WAS.serverName=IRAppSrv
WAS.profile=
# WAS installable apps directory is set in the Main
WAS.installableAppsDir=

# -----
# Variables used for WAS configuration steps
# -----

# The name of the application to be installed
WAS.appName=Flight400

# the install EAR or WAR file
WAS.appInstallFile=Flight400.ear

# The name of the IHS server instance to be started,
IHS.webServerName=IRHTTP
```

Script directory

It is a good practice to keep the scripts in your application project. So, we create a new folder called Flight400_Scripts to store all the custom scripts for our application wrapper (see Figure 6-71).

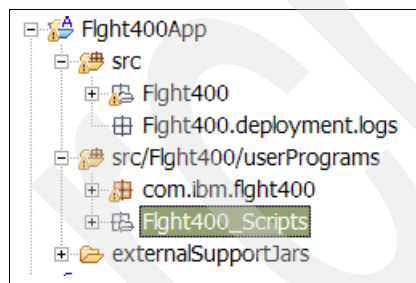


Figure 6-71 The scripts folder

We import the script for installing the WebFacing application (developed prior to building a wrapper, see 6.4.1, “Before developing a wrapper” on page 152) into this folder. We also copy and modify the CheckAppInstall.jacl script (from the IRU2_1SampleI5OS project) to verify if the Flight400 application is already installed on the system.

The final step in developing the user programs is to update the application.xml file. We add information on the Variables, Files, and Libraries tabs (see Figure 6-72, Figure 6-73, and Figure 6-74).

The screenshot shows the 'Application Variable Information' dialog box with the 'Variables' tab selected. The dialog is titled 'application.xml' and has a tab bar at the bottom with 'Welcome', 'General', 'Programs', 'Variables', 'Files', 'Libraries', and 'Source'. The 'Variables' tab is active, showing a list of application variables on the left and configuration options on the right.

Application Variables

- String Variable: appSvrName
- String Variable: httpServerName

Buttons: Add..., Remove, Up, Down

Basic Variable Configuration

Specify label and help text for the selected variable.

- *Name: appSvrName
- *Label text: Application Server name
- Help text:
- Type: Typical variable

Variable Associations Configuration

Add, remove or edit the associations for the selected variable.

- Property Association: appSvrName
- CID File Association: WAS.serverName

Buttons: Add..., Remove

Variable Validation Configuration

Specify validation requirements for the selected variable. You may also specify a default value.

- Default value: IRAppSvr
- Required: ☒
- Make uppercase: ☐
- Make lowercase: ☐
- Minimum length: 1
- Maximum length: 100
- Validation rules: Invalid prefix: "0", Invalid prefix: "1", Invalid prefix: "2", Invalid prefix: "3", Invalid prefix: "4", Invalid prefix: "5"

Buttons: Add..., Remove

Figure 6-72 Variables tab

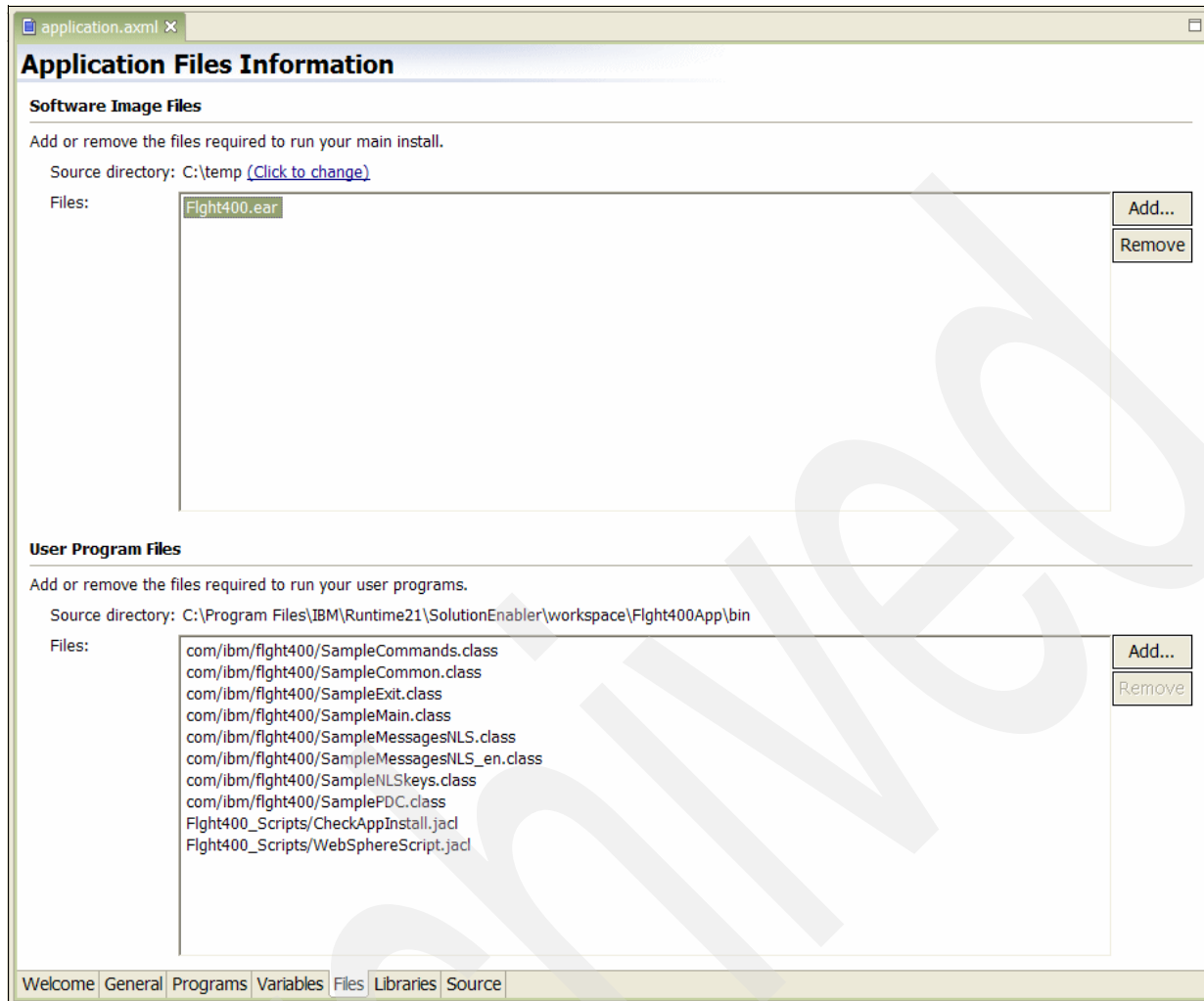


Figure 6-73 Files tab

The jt400Native.jar file contains classes required to access iSeries and execute commands on the system (see Figure 6-74).

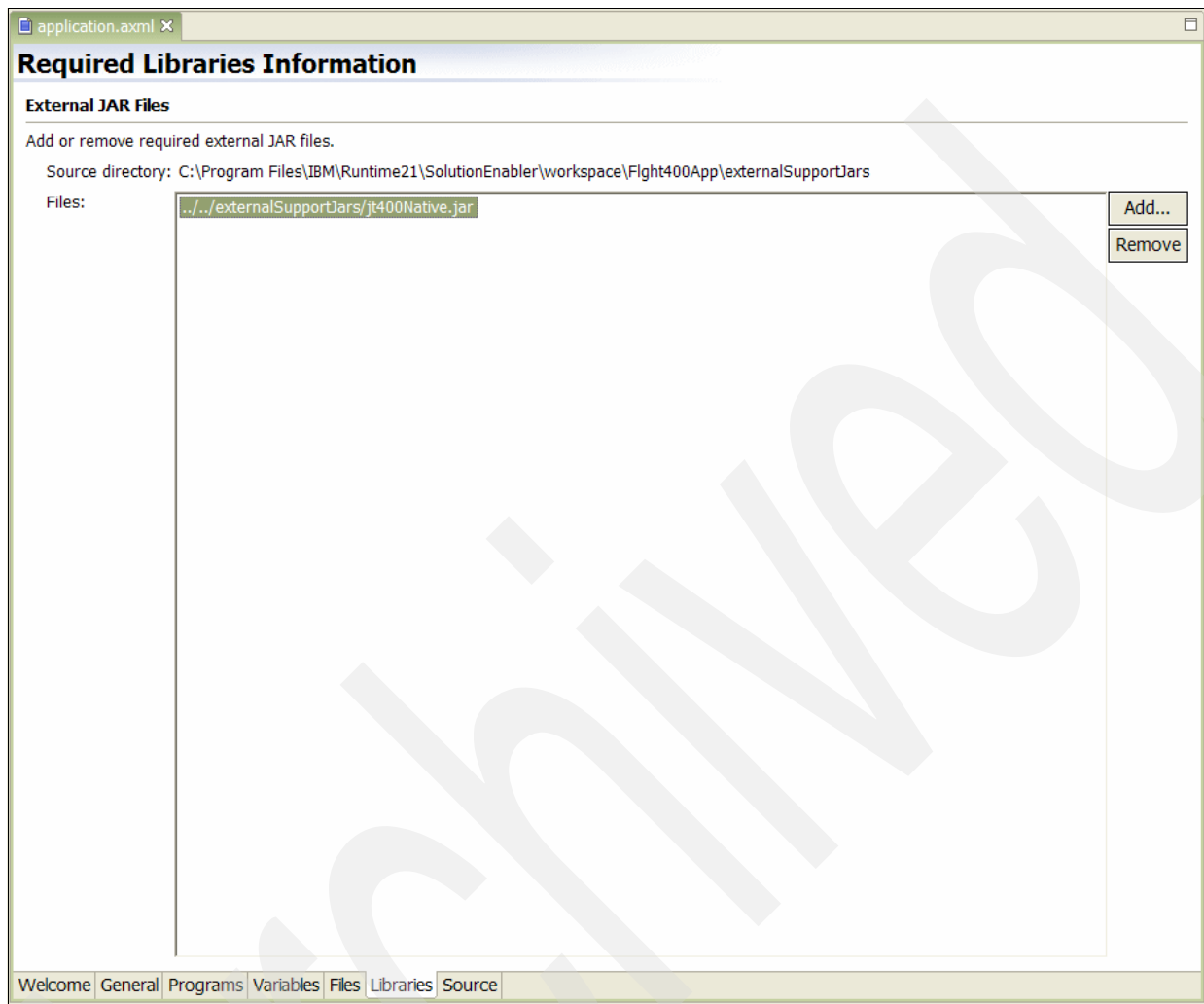


Figure 6-74 Libraries tab

6.4.4 Creating the solution project

Now we are ready to create a solution based on the Flight400 application. We create the solution project in the same way as it was described in 6.2.5, "Creating Trade6 solution project" on page 132.

To run the Flight400 application we need to configure the HTTP server and IBM WebSphere® Application Server - Express Version 6 for iSeries. It's possible that WebSphere is not installed on iSeries, so we need to have the WAS installation task as part of our solution.

Figure 6-75 shows what tasks we include in our solution.

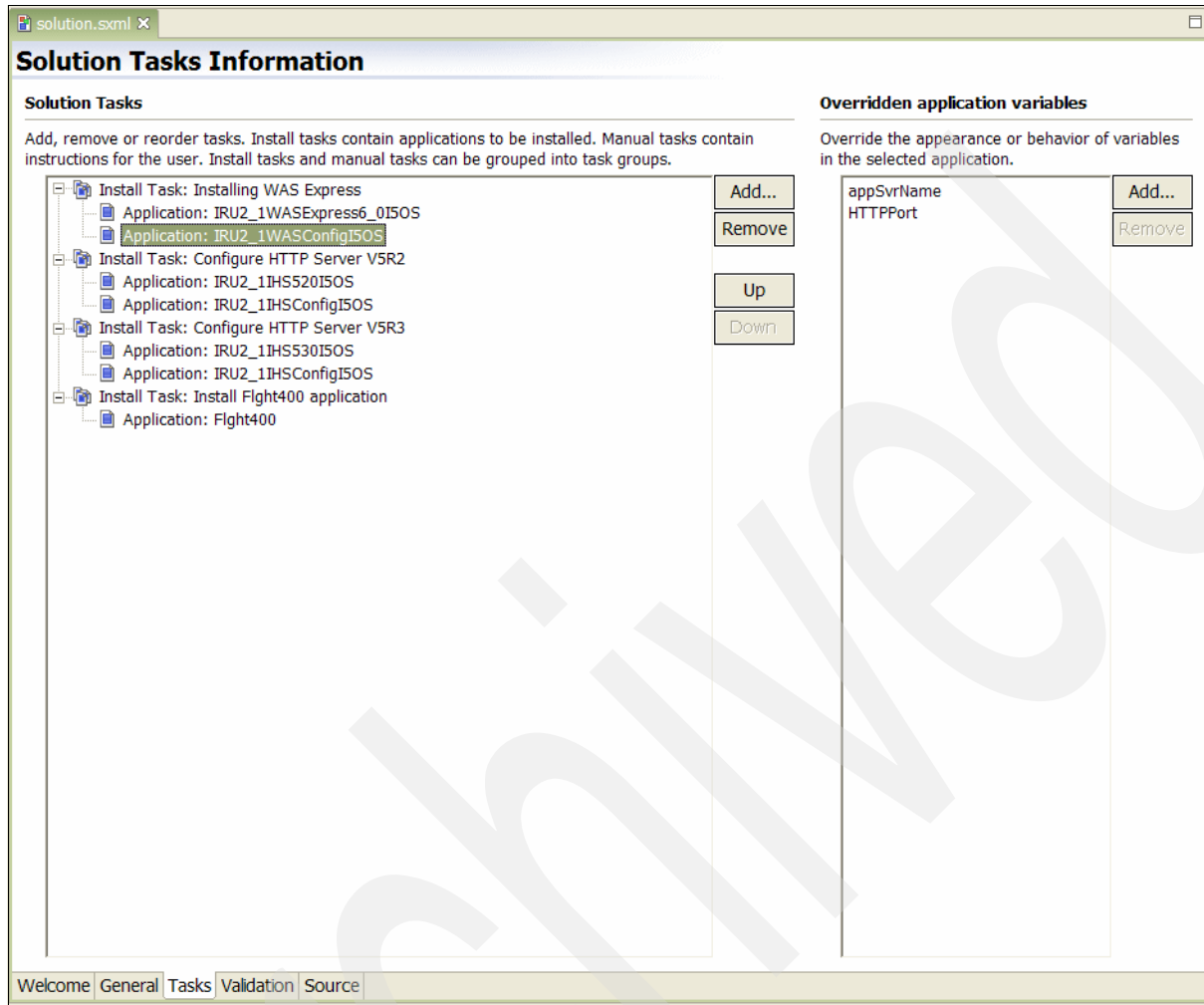


Figure 6-75 Tasks in the solution

We also add three shared variables for HTTP port, HTTP, and WAS server names (see Figure 6-76).

Figure 6-76 Shared variables validation information

At this point we have developed all the pieces of the solution and we are ready for testing.

6.4.5 Generating and testing the solution

After we complete the development part, we must generate the application and solution projects and the deployment packages. Instructions for generating the solution are given in 6.2.6, “Building the Trade6 solution” on page 143.

Next, we install the IBM Installation Agent (IIA) on iSeries and start it (see 8.4.2, “Installing on i5/OS” on page 190). When the IIA is running, we can start the remote deployment of our solution. See 8.5.3, “Scenario 3: Deploying to the remote systems (two systems)” on page 209, for the instructions on remote deployment.

6.5 Debugging user programs

The Express Runtime Developer tool provides a utility to debug user programs. This comes in handy to make sure that you get the right results in your wrappers.

The following actions are required to debug a solution deployment using the Debugger utility:

- ▶ The Express Runtime Developer and Deployment Wizard must be installed on the same machine.
- ▶ The source files (wrappers and java source codes) for the solution and the application(s) for debug must be located in the Express Runtime Developer workspace.
- ▶ The applications must be written in Java.
- ▶ The solution is deployed on Windows, Linux, or the local host.

The following steps describe how to debug an application as it deploys:

1. Open the Express Runtime Developer by selecting **Start → Programs → IBM Express Runtime 2.1 → Express Runtime Developer**.
2. Select the application project and Java program you wish to debug from Package Explorer. For instance double-clicking **TradeWinMain.java** in the application project.
3. On the program source code, bring your cursor to the line where you wish to suspend execution. Right-click the left margin of the Java code and select **Toggle Breakpoint** (see Figure 6-77). Alternatively, you can double-click the left margin on the line where you want to place a breakpoint.

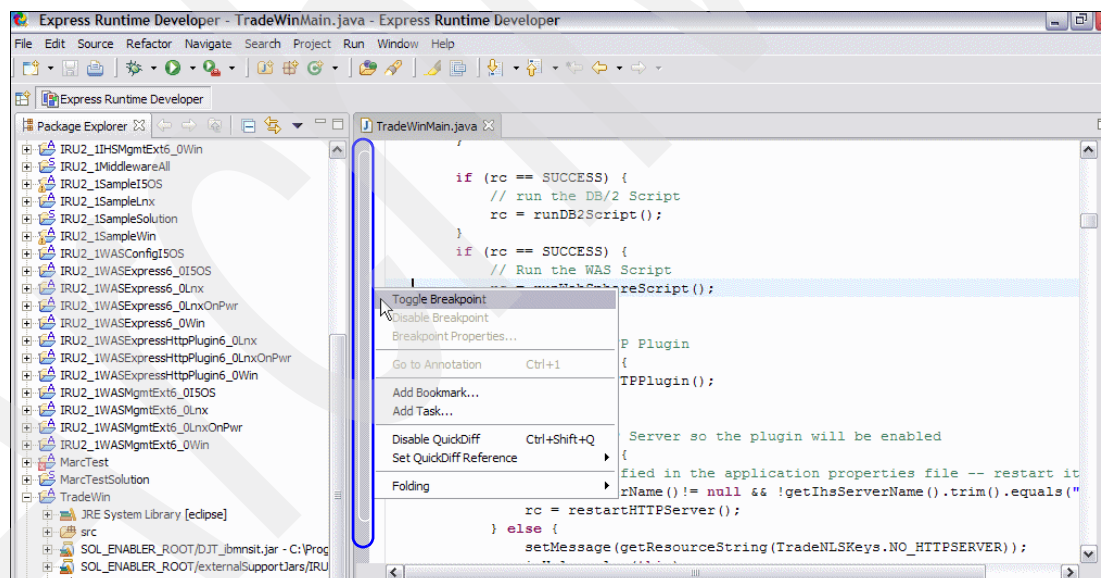


Figure 6-77 Toggle Breakpoint

4. The result shows a symbol highlighting the line with a breakpoint (see Figure 6-78).

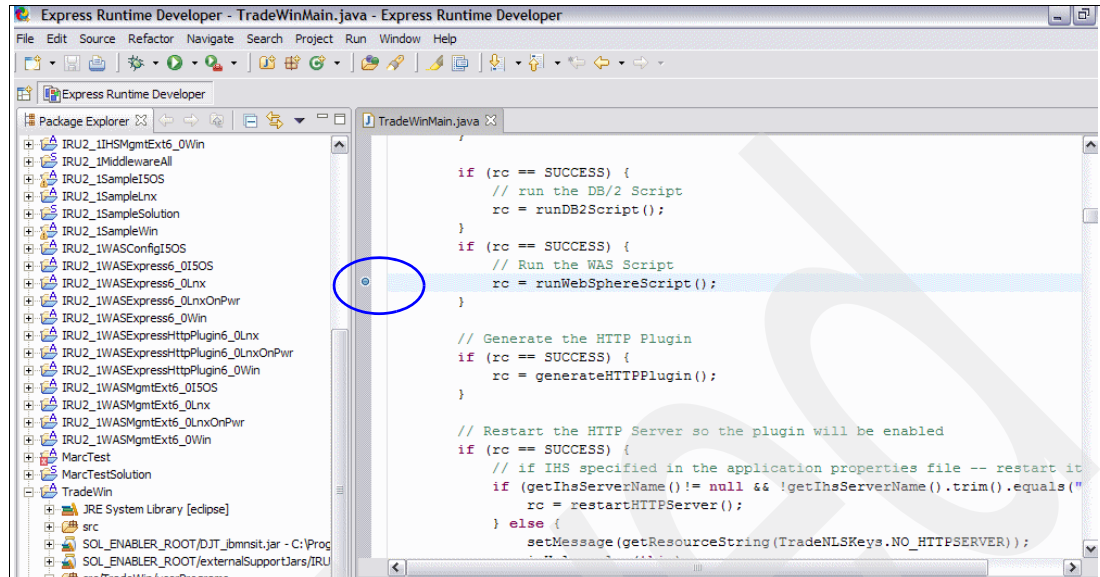


Figure 6-78 Breakpoint symbol

5. Run the solution in debug mode. In Package Explorer right-click the solution project (in our example **TradeWinSolution**), and choose **Test in Deployment Wizard** (see Figure 6-79).

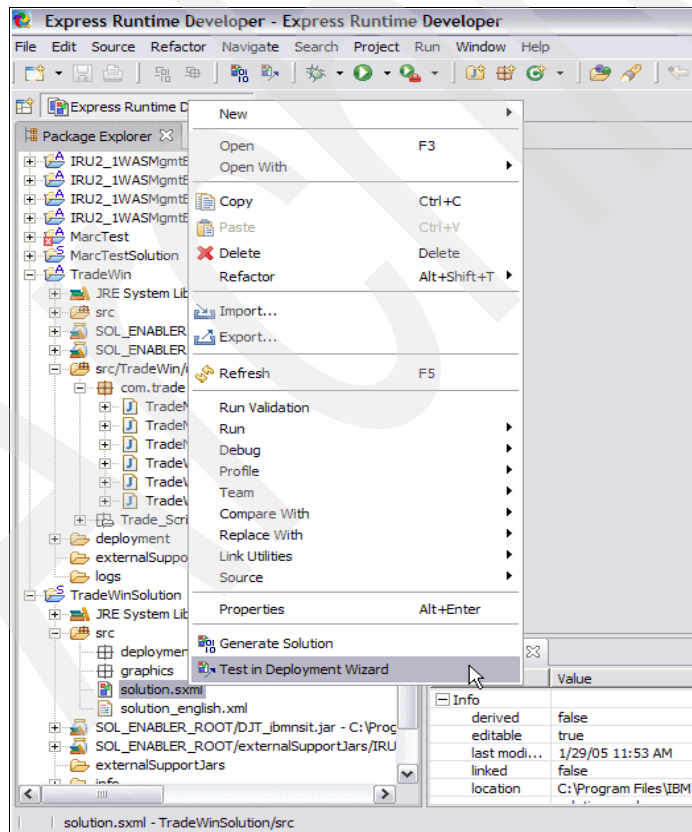


Figure 6-79 Test in Deployment Wizard

Note: You may be prompted to build the solution. Click **OK** to generate the application.

6. When the deployment wizard appears, configure and deploy the application you wish to debug. During the deployment, if the line you set a breakpoint on is executed, execution should stop, and control is transferred to the Express Runtime Developer. Using the standard eclipse debugging tools, you can now debug the program as it runs (either locally, as part of a local deploy, or remotely).

Note: When a breakpoint has been reached, the program will not continue processing until **Resume** is selected in Express Runtime Developer.

For more information on using this feature, refer to Chapter 21 in *Rational Application Developer V6 Programming Guide*, SG24-6449. This book is about Rational Application Developer, but the debugging techniques it describes are identical to the Rational Web Developer techniques.

Packaging a solution

In the previous chapter, we successfully generated the solution project. With the JAR and serialized files in place for our solution, we can package our solution in the form suitable for deployment.

This chapter includes the following topics:

- ▶ Creating the solution package for Trade6 on Windows
- ▶ Creating the solution package for WebFacing
- ▶ Creating installation CDs for a solution
- ▶ Packaging a solution with just the application

7.1 Creating the solution package for Trade6

The final step in making the solution ready for deployment is exporting the solution. This requires that the application and solution projects must be built first. For the instructions on how to generate the application and solution projects, refer to the section “Building the Trade6 solution” on page 143.

To start exporting Trade6, follow these steps:

1. Open Express Runtime Developer by selecting **Start → Programs → IBM Express Runtime 2.1 → Express Runtime Developer**.
2. From Package Explorer, right-click the solution project **TradeWinSolution** and select **Export** (see Figure 7-1).

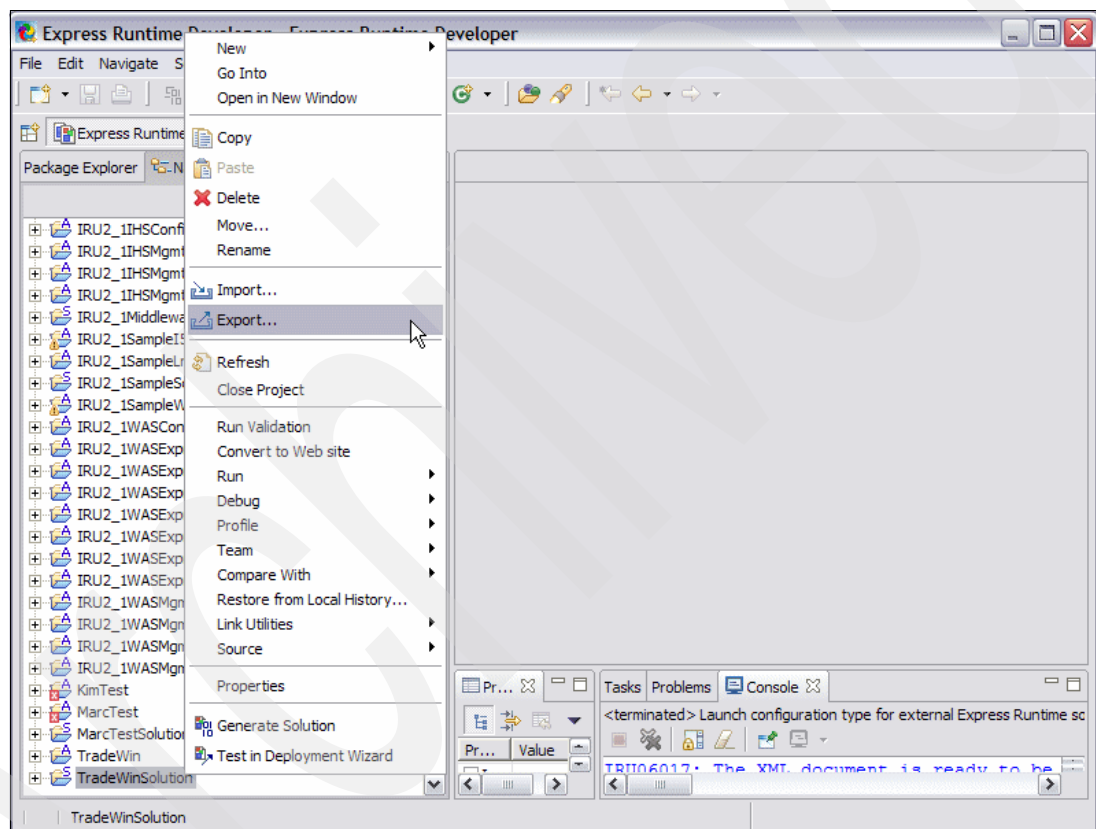


Figure 7-1 Export solution option

3. On the Export window, select **Express Runtime Solution** and click **Next** (Figure 7-2).

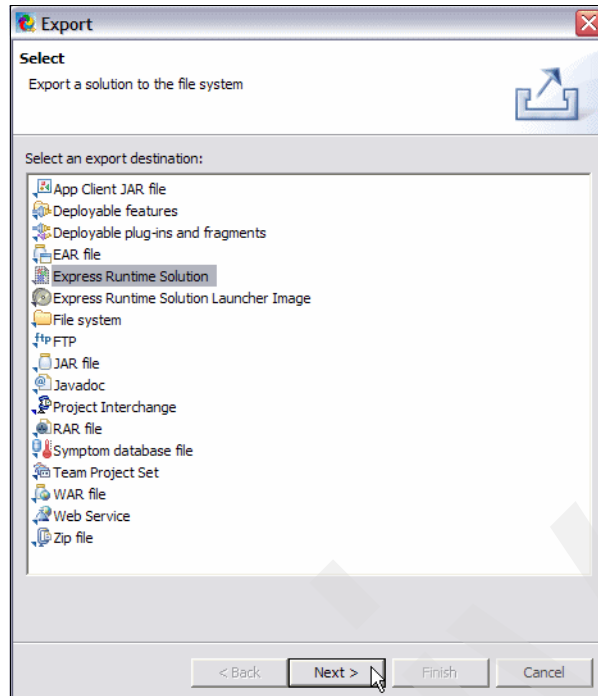


Figure 7-2 Selecting Express Runtime Solution

4. Select and/or enter the following values and click **Finish** (see Figure 7-3):

- Project to export: **TradeWinSolution**
- To directory: C:\Program Files\IBM\Runtime21\SolutionEnabler

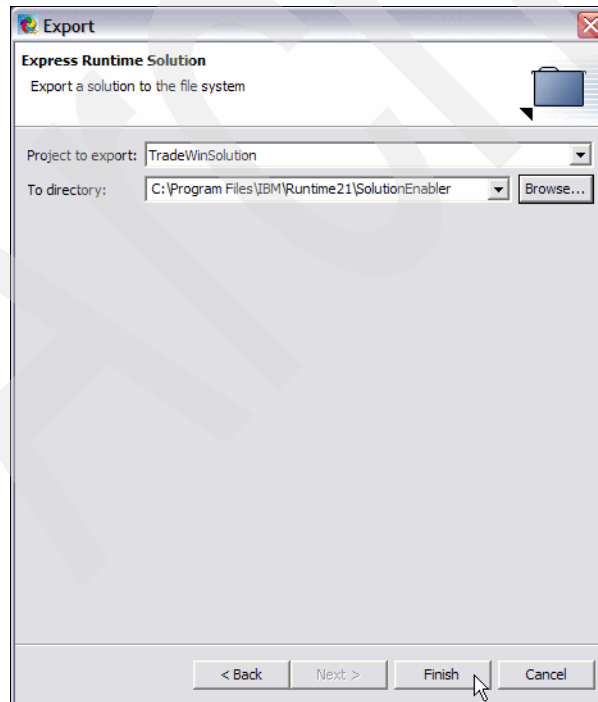


Figure 7-3 Selecting project to export

5. You are prompted to replace the existing solution .ser file and associated JAR files used with this solution. Click **Yes to All** to replace existing exported solution packages.

Now that we have completed building (6.2.6, “Building the Trade6 solution” on page 143) and exporting Trade6 successfully, the following files comprise our package of the Trade6 solution:

- ▶ TradeWinSolution.ser. This file located in *<ER install dir>\SolutionEnabler*.
- ▶ The following software images:
 - tradewin.xx.userPrograms.jar. This contains our application’s user programs. This file includes the class files for the Trade6 user programs and the content of the Trade_ScriptDir folder.
 - iru2_1db2express8_2win.xx.userPrograms.jar
 - iru2_1wasexpress6_0win.xx.userPrograms.jar
 - iru2_1wasexpresshttpplugin6_0win.xx.userPrograms.jar
 - iru2_1ihs6_0win.xx.userPrograms.jar

These are located in the following directory:

<ER workspace dir>\SolutionEnabler\userPrograms

- ▶ tradewin.xx.jar is located in the directory *<ER install dir>\IRU_common_resources\mediaJars*. This file contains the trade.ear file.

We can now deploy Trade6 on the same workstation with these files.

In deploying Trade6 using a staging server, our package needs only the following files:

- ▶ TradeWinSolution.ser
- ▶ tradewin.xx.userPrograms.jar
- ▶ tradewin.xx.jar

Note: The middleware JAR files are already present in the staging server. Hence, there is no need to include them in the Trade6 package.

For more information on deployment, refer to Chapter 8, “Deploying a solution” on page 177.

7.2 Creating installation CDs for a solution

The following steps can be performed in order to make the installation CDs for a solution. The example below assumes the solution is on the local host. IBM Installation Agent is not required:

1. Open Express Runtime Developer by selecting **Start** → **Programs** → **IBM Express Runtime 2.1** → **Express Runtime Developer**.
2. From Package Explorer, right-click the **TradeWinSolution** folder and select **Export**.
3. On the Export window, select **Express Runtime Solution Launcher Image** and click **Next** (Figure 7-4).

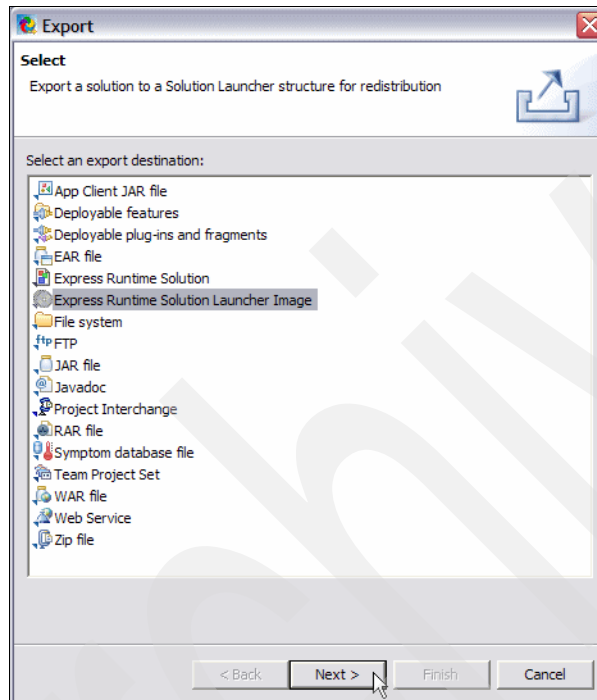


Figure 7-4 Select Express Runtime Solution Launcher Image

4. On the Express Runtime Solution Launcher Image window, use the following values, then click **Next** (Figure 7-5):
- Project to export: **TradeWinSolution**
 - Media size (MB): **650**. You can use the value, which is appropriate for the size of your media (CD or DVD).
 - Destination directory: C:\TradeWinCDImage
 - Solution Launcher operating system: **Windows**
 - Install Location: C:\SolutionFiles
 - Temporary drive: C:\

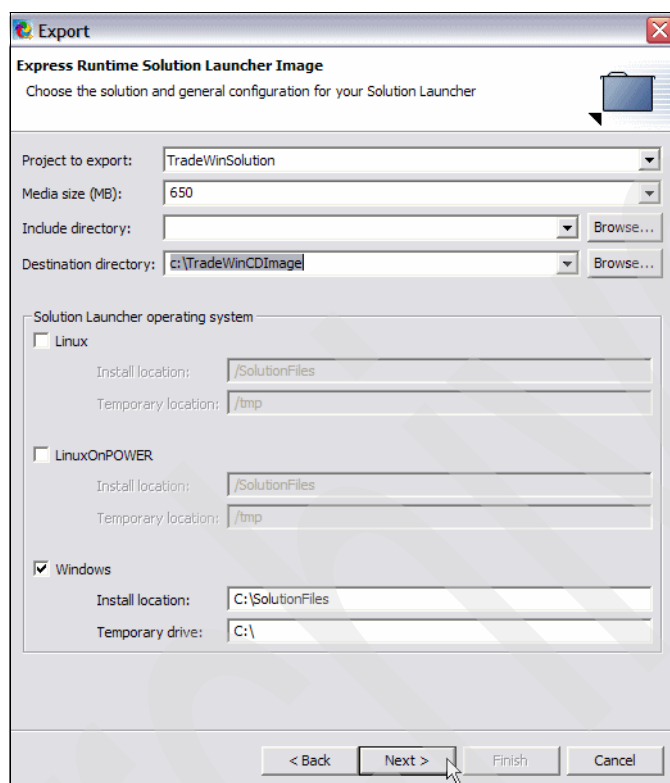


Figure 7-5 Express Runtime Solution Launcher Image

Note: For demonstration, we used C:\TradeWinCDImage in the Destination directory field. Feel free to change this when you export your own solution.

5. On the next window, enter the following values and click **Next** (Figure 7-6):

- Vendor name: IBM
- Vendor Web site: `www.ibm.com`

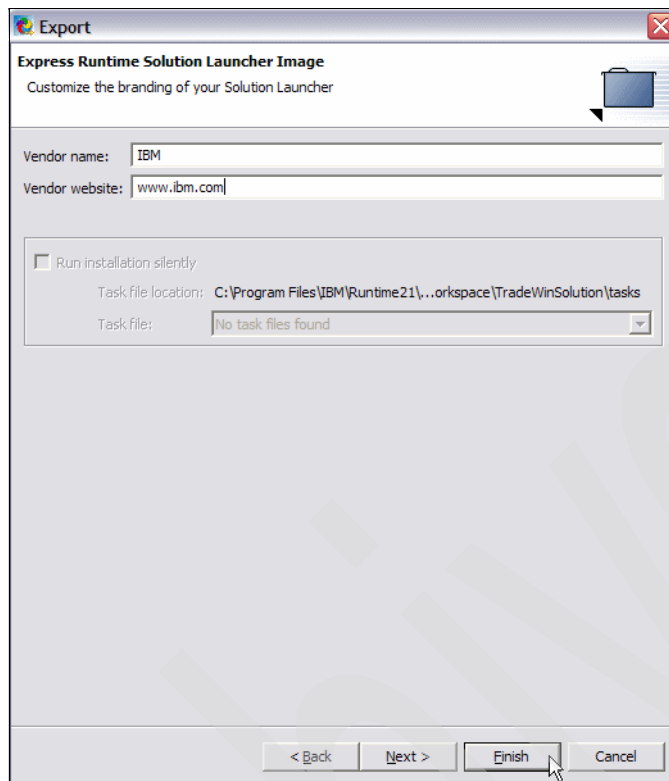


Figure 7-6 Enter vendor information

Note: If the install/target directory does not exist, a pop-up message will appear asking if the directory can be created. Click **Yes**.

6. In the next window, select **English** for Default Language and click **Next** (Figure 7-7).

Note: You need to select all supported languages on this panel.

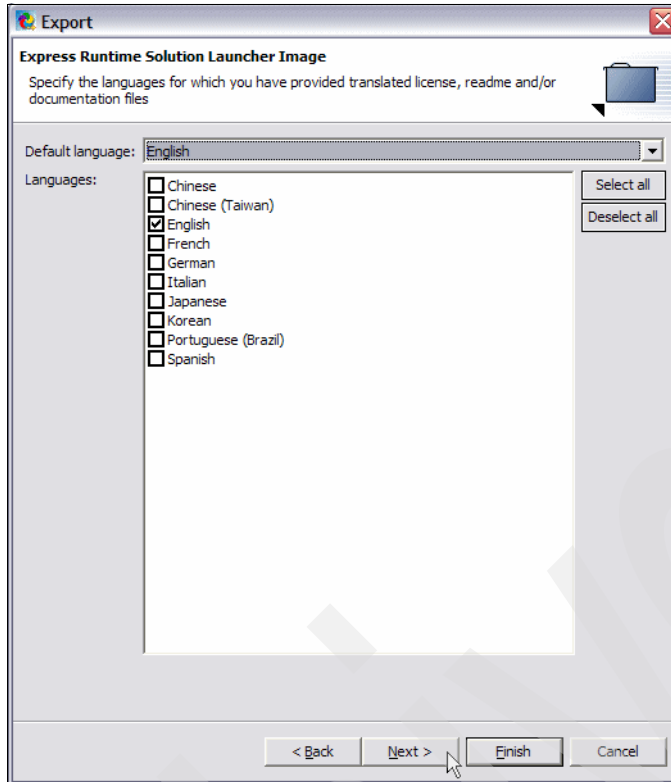


Figure 7-7 Export - Default Language

7. The next window enables you to configure licenses. Click **Next** (Figure 7-8).

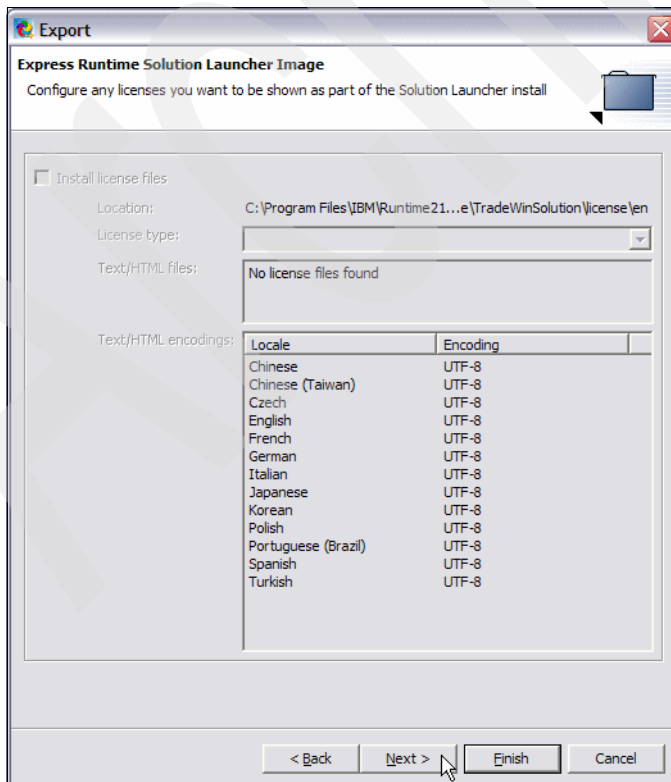


Figure 7-8 Export - Configure license

8. The next window allows you to place an image during solution installation. This can be used to display company logos or other emblems. In addition, this window also lets you put readme and documentation files related to the solution. Click **Finish** (Figure 7-9).

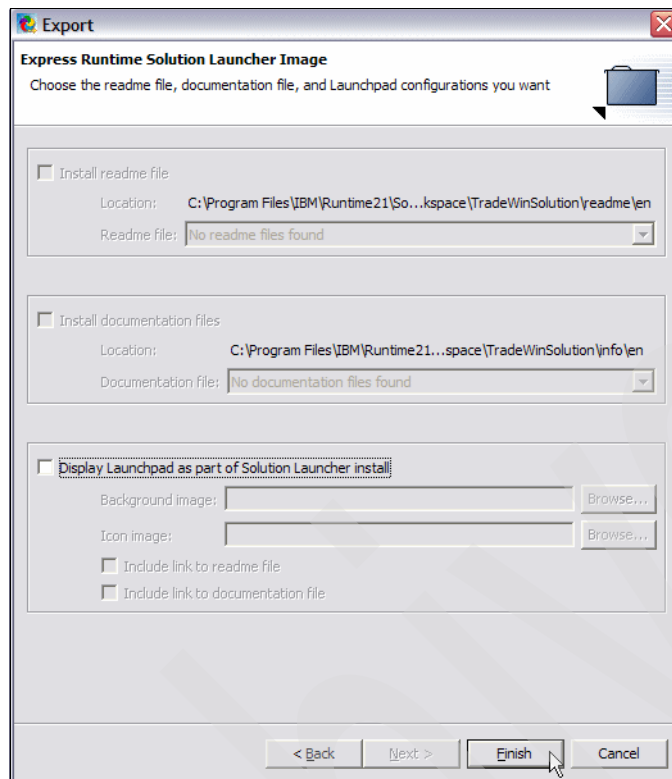


Figure 7-9 Export - Launchpad information

9. The Export window displays a progress indicator as shown in Figure 7-10.

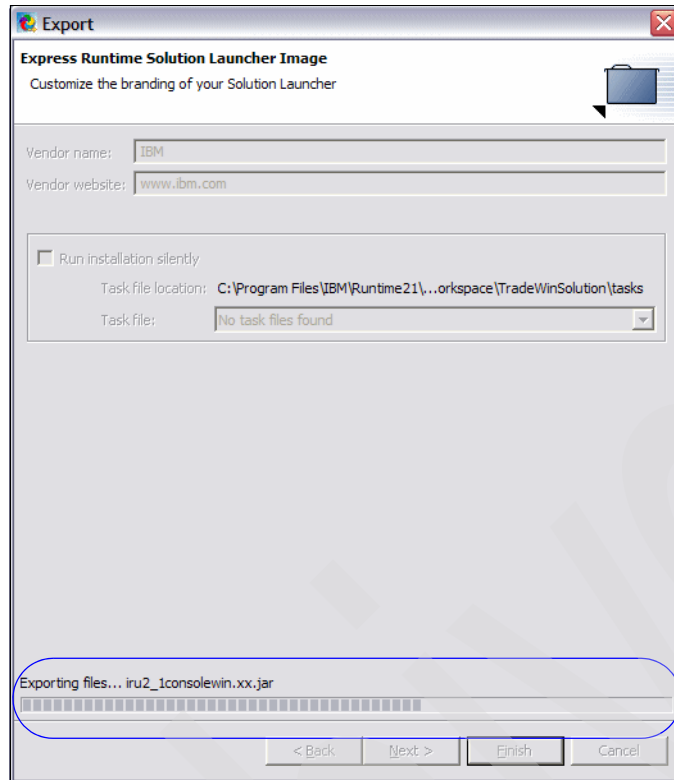


Figure 7-10 Export progress indicator

10. When the window closes, the export process has completed. Verify the content of the folder C:\TradeWinCDImage by using Windows Explorer (see Figure 7-11).

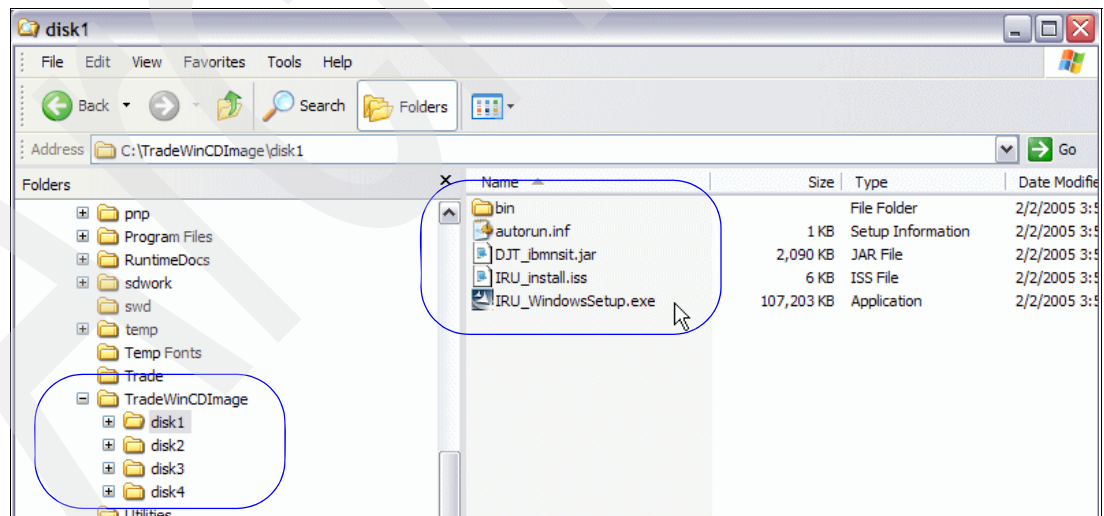


Figure 7-11 CD image files

Now you can burn the content of the TradeWinCDImage folder to CDs

7.3 Packaging a solution with just the application

In a scenario where there is a need to update an existing application (possibly to apply a new fix level or version), making a package of just the application is the best approach rather than including all the required middleware in a solution. This gives you the flexibility of not having to attempt middleware installation, leaving the current middleware configuration intact in your target environment.

Basically, the same steps provided in Chapter 6, “Developing a wrapper” on page 91 are used with some minor changes on the part of developing the solution project. You may use the same application .ser file. However, if you need to change some information in the application.xml and/or user programs of your project, the application project needs to be regenerated.

In the solution project, you can modify or create a new solution.xml with the same information as your existing version. To make the solution deploy just the application, ensure that your application (for instance, TradeWin) is the only application in the install task under the Solution Tasks section of the Tasks page (see Figure 7-12). On the Validation page, remove all variables that are not required during the deployment of your application. Rebuild the solution using the procedure in 6.2.6, “Building the Trade6 solution” on page 143.

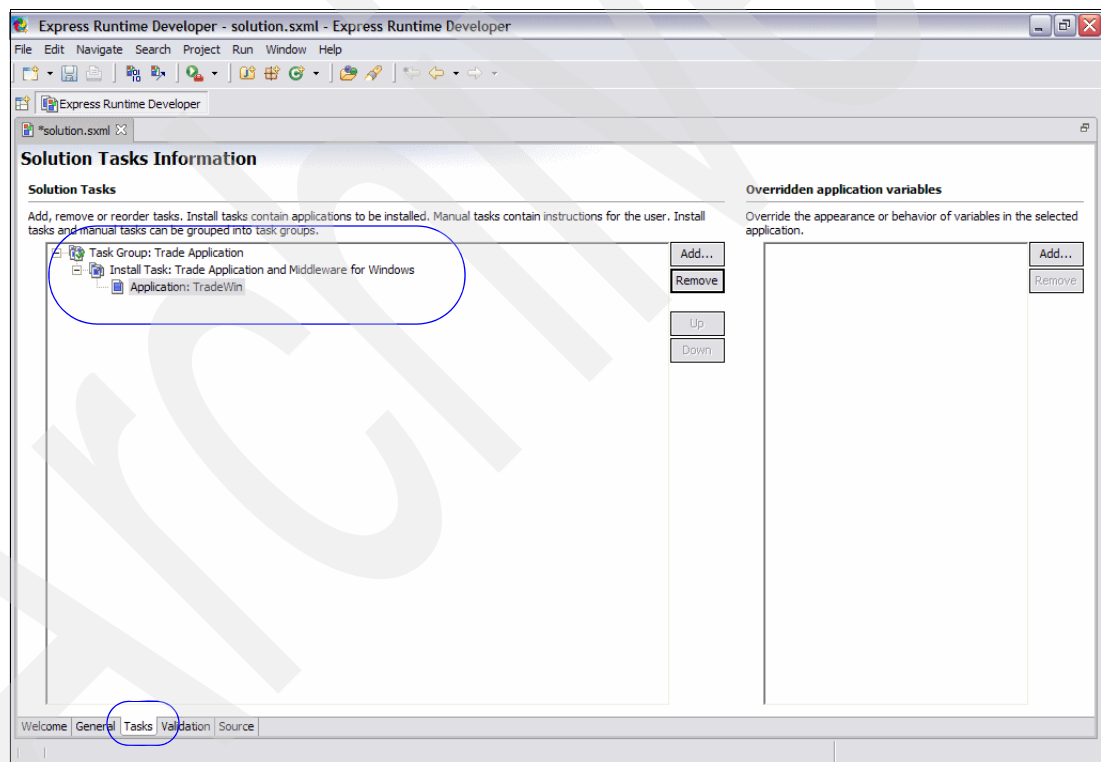


Figure 7-12 Install task for single application solution

This scenario produces the following files:

- ▶ TradeWinSolution.ser (application only)
- ▶ tradewin.xx.userPrograms.jar
- ▶ tradewin.xx.jar

Your user programs will check for all prerequisites, such as the existence of DB2 or WebSphere Application server on the target system, but if some middleware is not installed, your solution will not install it.

You may now make a package out of these files for deployment.

Deploying a solution

The goal of this chapter is to demonstrate how to deploy a solution. Deployment Wizard installs and configures the IBM middleware and applications in a single process. We highlight some key points during the deployment application process. This chapter also describes several deployment scenarios to cover most of the deployment possibilities.

This chapter discusses the following topics:

- ▶ Supported platforms
- ▶ Deployment methods
- ▶ Installing the IBM installation Agent
- ▶ Three deployment scenarios
- ▶ What if you have one of the components installed?
- ▶ Validating the deployment

8.1 Supported platforms

Express Runtime V2.1 supports several platforms from the deployment point of view. Microsoft Windows and Linux are supported for both activities of Development and Deployment.

Table 8-1 shows in detail which platforms are supported for which activities. In the column, Development or Deployment, we explain whether we can install on this platform's Express Runtime components. In the column, Deployable as Target Operating System, we state whether we can deploy on this platform.

Table 8-1 Express Runtime V2.1 Platform Table

Platform	Operating System	Development or Deployment	Deployable as Target OS?
Windows	Windows XP Professional SP 2	both	No
	Windows 2000 Server SP4	both	Yes
	Windows 2000 Advanced Server SP4	both	Yes
	Windows 2000 Professional SP3	both	Yes
	Windows Server 2003, Standard Edition SP1	both	Yes
	Windows Server 2003, Enterprise Edition SP1	both	Yes
Linux (Intel platforms only)	Red Flag Advanced Server 4.1	deployment only	Yes
	Red Hat Enterprise Linux 3.0 WS/AS/ES	both	Yes
	SUSE LINUX Enterprise Server 8.0	deployment only	Yes
	SUSE LINUX Enterprise Server 9.0	both	Yes
Linux (IBM POWER5 processor-based technology systems only)	SUSE LINUX Enterprise Server 8.0	deployment only	Yes
	SUSE LINUX Enterprise Server 9.0	deployment only	Yes
	Red Hat Enterprise Linux AS 3.0	deployment only	Yes
OS/400	V5R2	neither	Yes
i5/OS ¹	V5R3	neither	Yes

¹ The OS/400 operating system is known as the i5/OS operating system beginning with V5R3.

8.1.1 System requirement

We can say that during the deployment process, we install a cohesive set of middleware components and ISV's applications in a single process on several target systems. So the system requirement for the Deployment process is the same as the system requirements for every middleware component.

Distributed platforms

For Distributed platforms, we refer to Windows and Linux systems. For requirements for both of these platforms, we refer to any middleware component.

Windows

Windows system requirements are:

- ▶ A minimum of 512 MB of memory. However, 1 GB is recommended.
- ▶ A Pentium III-class processor with a minimum clock speed of 600 MHz. However, a Pentium IV class processor with a minimum clock speed of 1.2 GHz is recommended.
- ▶ An SVGA monitor with a minimum 1024 x 768 video resolution configured to display a minimum color depth of 256 colors (only required for GUI-based tasks, such as interactive installation; not needed for silent installation or command-line functions).
- ▶ A local area network (LAN) connection.
- ▶ The following network support, which must be configured when deploying solutions to network-attached target computers:
 - TCP/IP
 - DNS
- ▶ Internet Explorer 6.0 SP 1+ Web browser to view the online documentation and readme.
- ▶ Approximately 75 MB of disk space to install the IBM Installation Agent, plus 75 MB available in the system temporary directory. Additional space is required to deploy and run each application that you install on a target computer.

Linux on Intel

Linux Intel system requirements are:

- ▶ A minimum of 512 MB of memory; 1 GB is recommended.
- ▶ At minimum, an Intel Pentium III class processor with a minimum clock speed of 600 MHz. A Pentium IV class processor with a minimum clock speed of 1.2 GHz is recommended.
- ▶ An SVGA monitor with a minimum 1024 x 768 video resolution configured to display a minimum color depth of 256 colors (only required for GUI-based tasks, such as interactive installation; not needed for silent installation or command-line functions). Any Linux supported video card that supports the resolution requirements.
- ▶ A local area network (LAN) connection.
- ▶ The following network support must be configured when deploying solutions to network-attached target computers:
 - TCP/IP
 - DNS
- ▶ Mozilla 1.4 Web browser to view online documentation and readme.
- ▶ Approximately 85 MB of disk space to install the IBM Installation Agent, plus 85 MB available in the system temporary folder. Additional space is required to deploy and run each application that you install on a target computer.

Linux on POWER

Linux on POWER system requirements are:

- ▶ 2 GB of memory.
- ▶ A RS64-IV processor with a minimum clock speed of 600 MHz.
- ▶ An SVGA monitor with a minimum 1024 x 768 video resolution configured to display a minimum color depth of 256 colors (only required for GUI-based tasks, such as interactive installation; not needed for silent installation or command-line functions). Any Linux supported video card that supports the resolution requirements.
- ▶ A local area network (LAN) connection.

- ▶ The following network support must be configured when deploying solutions to network-attached target computers:
 - TCP/IP
 - DNS
- ▶ Mozilla 1.4 Web browser to view online documentation and readme.
- ▶ Approximately 85 MB of disk space to install the IBM Installation Agent, plus 85 MB available in the system temporary folder. Additional space is required to deploy and run each application that you install on a target computer.

Additional information

For details about system requirements for DB2 UDB, you can visit:

<http://www-306.ibm.com/software/data/db2/udb/dwe/sysreqs.html>

For details about system requirement of WebSphere Application Server and IBM HTTP Server, you can visit:

<http://www-306.ibm.com/software/webservers/appserv/doc/latest/prereq.html>

i5/OS

In this section we refer to i5/OS and OS/400 as OS/400.

For OS/400 systems, we also need a remote Windows or Linux machine to perform installation of the IBM Installation Agent. For this platform, the requirements are as follows:

- ▶ The user performing the installation or running the IBM Installation Agent on an i5/OS (known as OS/400 prior to Version 5 Release 3) must be a SECOFR user with the following permissions:
 - ALLOBJ
 - SAVSYS
 - JOBCTL
 - SERVICE
 - SPLCTL
 - SECADM
 - AUDIT
 - IOSYSCFG
- ▶ IBM OS/400 Version 5 Release 2 with Java group PTF SF99169 or i5/OS Version 5 Release 3 with SF99269.
- ▶ Host Servers (5722SS1, option 12).
- ▶ QShell (5722SS1, option 30).
- ▶ Java Developer Kit 1.4 (5722JV1, option 6).
- ▶ Crypto Access Provider 128-bit (5722AC3).
- ▶ The most recent WebSphere version 6 group and cumulative PTFs.
- ▶ The following network support, which must be configured:
 - TCP/IP
 - DNS
- ▶ A LAN connection.

- ▶ Disk space requirements:
 - On i5/OS or OS/400 systems:
Approximately 20 MB of space to install the IBM Installation Agent, plus 80 MB available in the system temporary folder.
 - On remote Windows or Linux machines:
Approximately 80 MB of space available in the system temporary folder for remote installations. A minimum video resolution of 1024 x 768 is required (only required for GUI-based tasks, such as interactive installation; not needed for silent installation or command-line functions).

Note: i5/OS is the platform integrated with DB2 UDB and IBM HTTP Server products. For this reason, Express Runtime V2.1 provides only the application wrapper for IBM HTTP Server for iSeries to configure this product on the system.

For more details about OS/400 system requirement, you can visit:

<http://publib.boulder.ibm.com/infocenter/iseries/v5r3/ic2924/index.htm>

8.2 Terminology used in this chapter

Before we discuss the deployment capabilities of Express Runtime V2.1, we need to define several terms as they apply to information in this chapter:

- ▶ A *Development System* is the system where we develop our applications and/or solutions. This system also contains the deployment environment (Deployment Wizard).
- ▶ A *Staging Server* is a system with only deployment environment. This box can be used for local or remote deployment.
- ▶ A *Target System* is a system where you deploy your solution.
- ▶ *Deployment Wizard* is a special program that guides you through the process of installing a solution to a target system. It simplifies the deployment process, making it really easy for the users.
- ▶ *Deployment tasks* are the tasks for installing middleware and applications on the target computers.
- ▶ *IBM Installation Agent (IIA)* is a special program that runs on the remote target system. It is used to support the secure communication with the staging server and execute the deployment tasks on the target system.

8.3 Deployment methods

There are two main methods to deploy a solution to a target system:

- ▶ From a staging server
- ▶ From CDs/DVDs

In each of these methods you can deploy a solution either locally or remotely. Figure 8-1 shows a diagram of possible deployment methods.

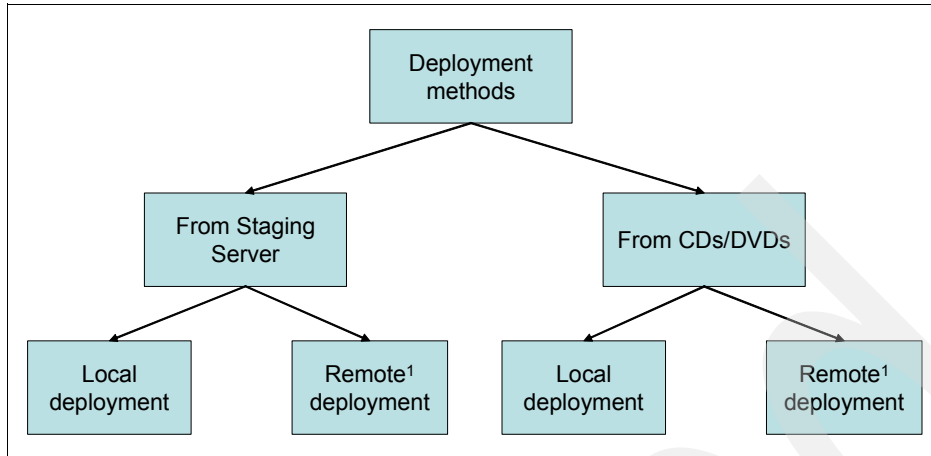


Figure 8-1 Deployment methods

¹You can deploy your solution to one or more systems.

Deploying from a staging server

A staging server contains Deployment Wizard and all files necessary to install your solution. A development system can be considered as a staging server because it has Deployment Wizard installed (during standard installation) as well as all the solution files created on this system.

As alternative, you can select a separate system and install only Deployment Wizard. You need to copy the solution files to this system (see 7.1, “Creating the solution package for Trade6” on page 166) in a separate step.

Deploying from CDs/DVDs

The second deployment method is to use CDs or DVDs that contain all components for your solution deployment. You create CD or DVD images using the export wizard in Express Runtime V2.1 (see 7.2, “Creating installation CDs for a solution” on page 169). The deployment process is done in three stages:

1. Solution Launcher is used to start the process.
2. It temporarily installs Deployment Manager to the system.
3. Deployment Wizard starts the deployment process. CDs/DVDs contain all the components, including the middleware images.

If you decide to abort the deployment, Deployment Wizard will be removed from your system.

Local and remote deployment

No matter what installation method you choose, you need to decide if a remote deployment will be supported. The decision to use remote deployment adds additional requirements for your deployment environment: You need to install IBM Installation Agent (IIA) on all target systems. See the next section for more information.

8.4 Installing the IBM Installation Agent (IIA)

Remote deployment is a process that installs a solution on a remote computer. We call this remote computer Target Computer. Before starting remote deployment, we need to install IBM Installation Agent on a Target Computer. The IBM Installation Agent is an application that acts as an interface for software installations. We need to install this agent on each remote computer where we want to deploy a solution.

The agent acts as a listener process that waits for a connection from a system where you run Deployment Wizard. On i5/OS (known as OS/400 prior to Version 5 Release 3)*, the IBM Installation Agent runs as a background program.

For supported platforms, refer to Table 8-1 on page 178. For system requirements, refer to 8.1.1, “System requirement” on page 178.

Note: Remember, for all platforms, additional space is required to deploy and run each application that you install on a target computer. Refer to IBM Installation Agent Product Documentation available on the First Steps menu for more details.

To locate the executable for the operating systems onto which you want to install IIA, refer to the following directory structure of the IIA installation CDs:

- ▶ **diskIIA1**
 - **doc/** — This directory contains Web-based documentation in any supported languages.
 - **readme/** — This directory contains installation instructions for IBM Installation Agent.
 - **linux/** — This directory contains installation programs and licenses for Linux.
 - **w32/** — This directory contains installation programs and licenses for Windows.
- ▶ **diskIIA2**
 - **linuxOnPOWER/** — This directory contains installation programs and licenses for Linux on POWER.
 - **os400/** — This directory contains installation programs and licenses for OS/400.

Errors encountered before the installation program is able to determine the destination path (or errors with the destination path specified) are logged to the current directory, unless write-access is not allowed. If write-access is not allowed, the errors are not logged.

To install the IBM Installation Agent, you must be logged on to the target computer as a user with administrator authority (root for Linux or Administrator for Windows, or a user profile that has SECOFR user class with all special authorities for i5/OS or OS/400). The installation fails if you are not logged on as an administrator, and there might not be any error messages logged to IRU_IIAInstall.log indicating the problem. If the installation fails and no information is available in the log file, exit the installation, log in as a user with administrator authority, and try the installation again.

8.4.1 Installing on Windows, Linux, and Linux on POWER

Follow these steps to install IIA:

1. You can perform installation of IBM Installation Agent with a graphical user interface (GUI). Select the command appropriate for your platform (these commands are located on different CDs and in different folders):

- For Windows, use this command:

IIA_WindowsLaunchpad.exe

- For Linux, use this command:

IIA_LinuxLaunchpad

- For Linux on POWER, use this command:

IIA_LinuxPPCLaunchpad

2. Ensure that both the setup.exe and launchpad.exe setup files exist in the same directory for each platform.

For Linux and Linux on POWER, we also need to have an X11 graphical environment available for using Launchpad program.

3. The LaunchPad main window contains the following links:

- **Install** — Starts the IBM Installation Agent installation wizard.
- **Installation Guide** — Displays the IBM Installation Agent information center, providing detailed information about installing the IBM Installation Agent.
- **Exit** — Exits the LaunchPad.

Click **Install** and follow the prompts to install the IBM Installation Agent as it is shown in Figure 8-2.



Figure 8-2 Install Shield Wizard for IBM Installation Agent

4. If, for any reason, you want change the temporary directory for your installation, you can perform the following commands from the command line on each platform instead of using the LaunchPad menu:

- For Windows use command `IIA_WindowsSetup.exe -is:tempdir <temp>`
- For Linux use command `IIA_LinuxSetup -is:tempdir <temp>`
- For Linux on POWER use command `IIA_LinuxPPCSetup -is:tempdir <temp>`

Where <temp> is the folder name we want to use as the temporary folder.

Skip this step if you are using the LaunchPad menu.

5. In the next panel, read and accept the license agreement. Click **Next**.
6. Select the destination path for this installation. A default path is displayed. Click **Next**.
7. In the next panel, specify a phrase for a secure key creation (Figure 8-3). This key will be used at connection setup to authenticate the request. At the deployment time, you will have to enter this phrase using the Deployment Manager interface to generate the secure key on the staging server.

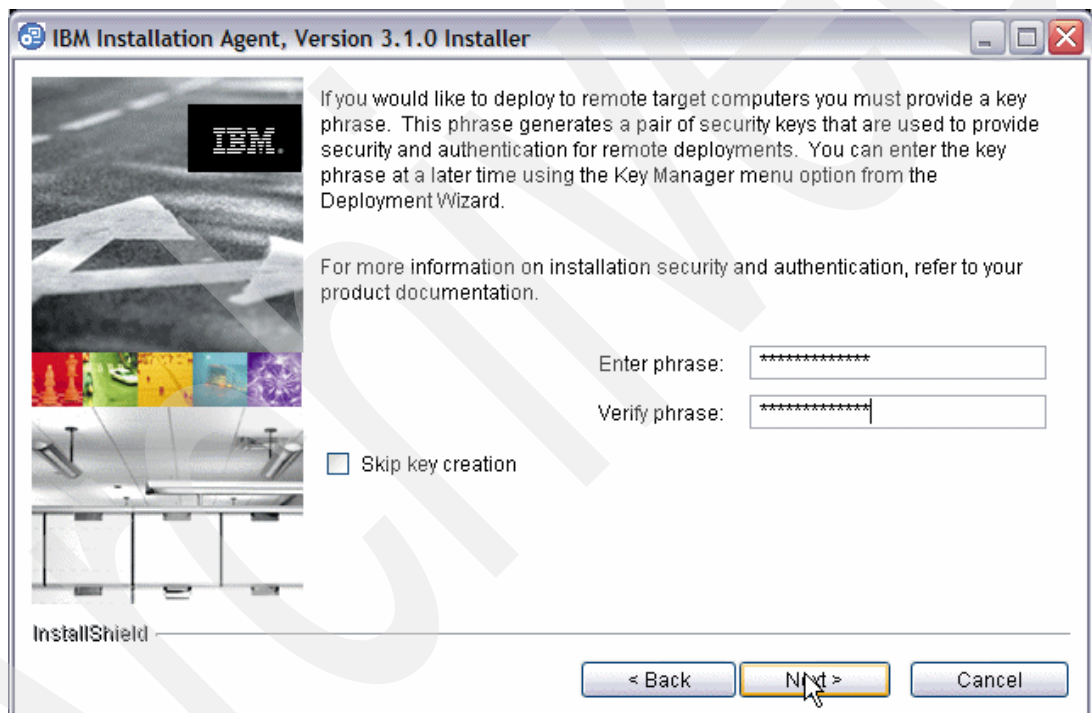


Figure 8-3 Generating secure key during installation of IBM Installation Agent

Note: You may skip this step by selecting **Skip key creation**. However, you have to create a key before starting the deployment. The installation creates the Key Manager shortcut that needs to be invoked to generate the key.

8. In the next panel:
- **Windows only.** Specify a user ID and password for the service listener. If the user ID does not exist, it is created as a member of the Administrators group. If the user ID already exists, it must be a member of the Administrators group.
- **Linux and Linux on POWER only.** Select the appropriate run levels for the IBM Installation Agent daemon.

9. When the installation has completed successfully, click **Finish**. You have a choice to launch the First Steps panel as shown in Figure 8-4.

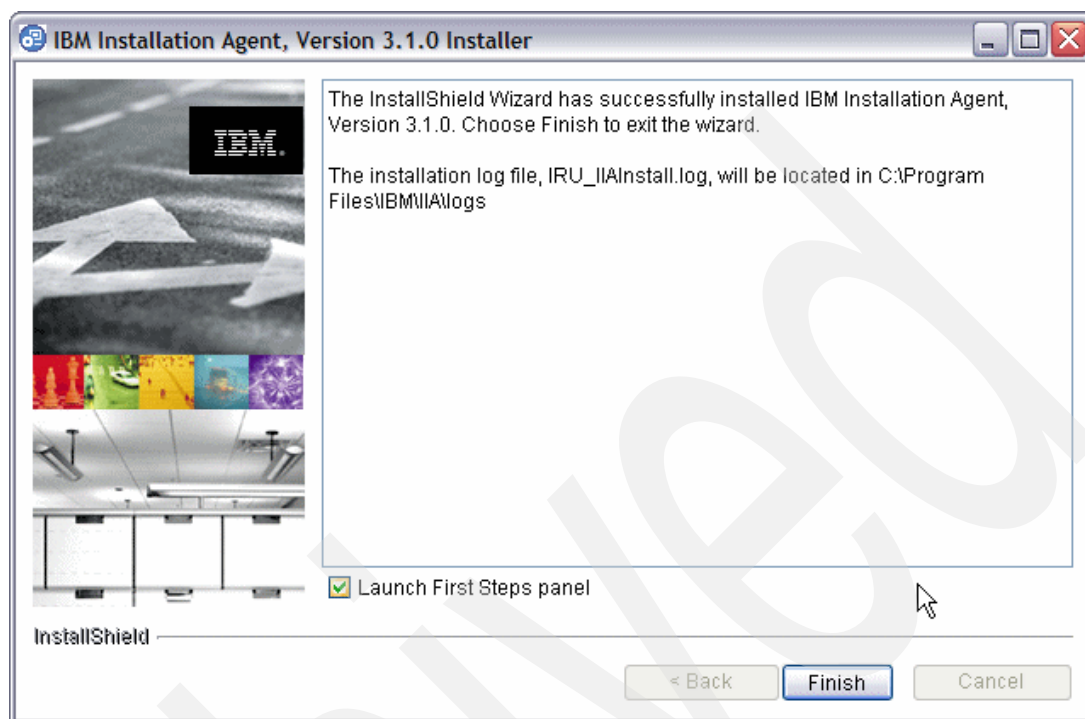


Figure 8-4 Completing LaunchPad Installation

Silent Installation of IBM Installation Agent

You can also perform a silent installation for every platform. There is a file called DJT_IIAsetup.iss in each platform directory. To perform a silent installation, enter the following command for each platform:

- For Windows, use the command:
`IIA_WindowsSetup.exe -options DJT_IIAsetup.iss`
- For Linux and Linux on POWER, use the command:
`IIA_LinuxSetup -options DJT_IIAsetup.iss`

Where DJT_IIAsetup.iss is a sample response file that you need to modify to perform a silent installation.

Example 8-1 shows this file for Linux platforms.

Example 8-1 Response file for silent installation

```
#Sample response file for the IBM Installation Agent V3.1.0 for Linux and Linux on POWER
#
# NOTE: NO spaces are allowed before the options (-P, -W, -G)
#
# Specify that the install should run silently.
#
-silent
#
# Key management provides security, control, and authentication for all
# installation requests. From the staging server, provide a phrase that will be
# used to generate a pair of security keys. From a target computer, enter this
# same phrase to allow the computer to communicate with the staging server.
```

```

# To create a key, uncomment out the next line and enter your key phrase.
#
#-W CreateKey.passPhrase="password"
#
# Provide an alternative install destination location.
# Default installation path: /opt/IBM
# To modify the default installation path, uncomment out the next line and
# insert a valid path for the location.
#
#-P product.installLocation="/opt/IBM"
#
# Stores the user's response to whether they want to replace a file that exists
# on their system. The possible values are: "yesToAll", "yes", "noToAll", "no"
# Note that this is for install action only.
#
-G replaceExistingResponse="yesToAll"
#
# Stores the user's response to whether they want to replace a file if that file
# is newer than the file being installed.
# The possible values are: "yesToAll", "yes", "noToAll", "no"
# Note that this is for install action only.
#
-G replaceNewerResponse="yesToAll"
#
# Stores the user's response to whether they want to create the destination
# directory if it does not already exist. The possible values are: "yes" and "no".
#
-G createDirectoryResponse="yes"
#
# Stores the user's response to whether they want to remove a file that exists
# on their system. The possible values are: "yesToAll", "yes", "noToAll", "no"
# Note that this is for uninstalls only.
#
#-G removeExistingResponse="yesToAll"
#
# Stores the user's response to whether they want to remove a file that has been
# modified since it was last installed.
# The possible values are: "yesToAll", "yes", "noToAll", "no"
# Note that this is for uninstalls only. Uncomment it for uninstall, otherwise,
# modified files won't be removed, and an exception will be logged in log file.
#
#-G removeModifiedResponse="yesToAll"
#
# Migrate previous version of IIA to the current version. The security keys from
# previous version will be preserved.
# Migrate to current version set: migration.navigationOptions=3
# Do not migrate to the current version set: migration.navigationOptions=4
#
-W migration.navigationOptions=3
#
# IIA service installation requires at least one run level on UNIX system.
# To view the various runlevels for this machine, go into the file /etc/inittab.
# Giving the level numbers seperated by commas.
#
-W startService.runLevels="3,5"
#
# If you wish to not install and launch the First Steps for the product then uncomment
# the following line and be sure that -P FirstSteps.active is set to False.
#
# -P FirstSteps.active=False

```

You can use this silent file as is. The installation will be performed with all default parameters. For instance, if you want generate a security key on the Target Computer during silent installation of IIA, you have to:

1. Uncomment the following row:

```
#-W CreateKey.passPhrase="password".
```

2. Change “password” with your key phrase as described in the comments for this parameter in the response file.

Validating installation of IBM Installation Agent

We can validate installation of IBM Installation Agent by checking the installation log file. For instance, on Linux systems, when silent installation completes, the pathname `/opt/IBM/IIA` is created on the system. The installation path `/opt/IBM` is the default path. We can modify it by editing the `DJT_IIAsetup.iss` response file. If we did not change any default pathname, we can find the log directory in `/opt/IBM/IIA/logs` and under it, the `/opt/IBM/IIA/logs/IRU_IIAInstall.log` installation log file.

Using IBM Installation Agent

If we skipped key creation during installation of IBM Installation Agent, we have to do that now before deploying the solution from the Staging Server. We must use the Key Manager shortcut to specify the phrase for generating the key needed for secure communication between the Staging Server and the Target Computer. We need to do that on both our systems: the Staging Server and the Target Computer. Furthermore, the phrase must be the same on both systems.

On Windows, we can find the Key Manager shortcut by clicking:

Start → Programs → IBM Installation Agent 3.1 → Key Manager

On Linux, we can find the Key Manager shell script at the following location:

`/opt/IBM/IIA/IRU_ia_start-key.sh`

Where `/opt/IBM/` is default installation pathname of IBM Installation Agent on Linux.

The IBM Installation Agent starts automatically when the target computer is started. On Windows, it runs as a service using the user ID you specified during installation. On Linux and i5/OS (known as OS/400 prior to Version 5 Release 3), it runs as a daemon using the run levels you specified at installation. A log file associated with the IBM Installation Agent is located in the `<install path>/IIA` directory with the name `IRU_IIA.log`.

We can also start or stop the agent from the command line:

- ▶ On Windows, we can start and stop it as a Windows Service.
- ▶ On Linux and Linux on POWER, we can start and stop IBM Installation Agent using the following shell scripts:

```
<install path>/IIA/IRU_ia_start-agent  
<install path>/IIA/IRU_ia_stop-agent
```

When the service of IBM Installation Agent is started, we can test if the connection between the Staging Server and the Target Machine is working fine. When we do that, a Transmission Control Protocol (TCP) socket is established between these two boxes. Communication occurs on a specific TCP Port (by default 1099). We can test if the Listener service on the target machine is running by using the command **netstat** as shown in Example 8-2 and Example 8-3.

Example 8-2 Netstat command from Windows command line

```
C:\Documents and Settings\Administrator>netstat -an | find "1099"
TCP    0.0.0.0:1099          0.0.0.0:0            LISTENING
C:\Documents and Settings\Administrator>
```

Example 8-3 Netstat command from linux prompt

```
[root@relinux IIA]# netstat -an | grep 1099
tcp      0      0 0.0.0.0:1099          0.0.0.0:*            LISTEN
[root@relinux IIA]#
```

Note: If there is a Firewall between the Staging Server and the Target Computer, you must open the TCP/IP Ports IBM Installation Agent:

- ▶ Listening port
- ▶ Data port
- ▶ Communication port

In this case you should define the ports when you start the deployment process. Figure 8-5 shows the Deployment Manager preferences window where you specify these ports.

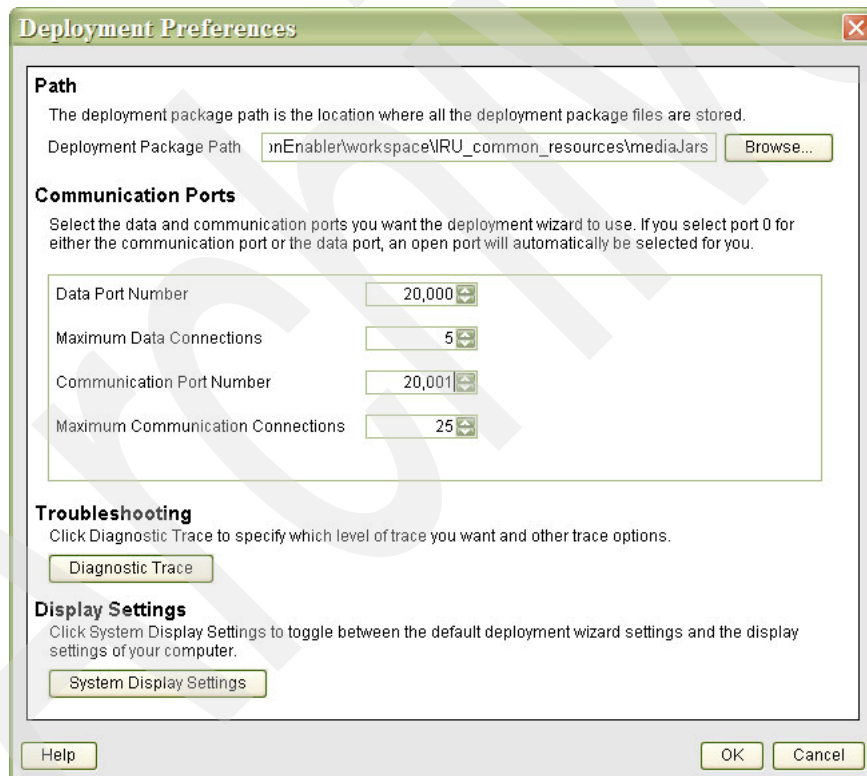


Figure 8-5 Selecting data and communication ports

8.4.2 Installing on i5/OS

While a Windows or Linux box is not required to install IIA on OS/400, it is certainly the easiest way of doing it. You run *IIA_OS400Setup* Java program. This Java program executes a remote installation of IBM Installation Agent on an OS/400 system. Now we describe a step-by-step installation of IBM Installation Agent on an OS/400 system using a Windows system as front-end.

1. Insert the diskIIA2 CD into CD-ROM and switch to the os400 directory.
2. Double-click **IIA_OS400Setup.exe**.
3. A logon popup is displayed on Windows front-end. Type a IP address or fully qualified name of OS/400 server where you want to perform installation, and supply a valid user name and password, as shown in Figure 8-6.



Figure 8-6 Sign on OS/400

Note: The signon window can hide behind other open windows on the desktop. For this reason, minimize all open windows before installing IIA.

4. After the setup program from the Windows front-end is connected on the OS/400 system and the username and password are verified, the InstallShield Wizard displays a popup window for language selection (see Figure 8-7). Select your language and click **OK**.



Figure 8-7 Select Language for installing IBM Installation Agent on OS/400

5. The Welcome window is displayed. Click **Next**.

Note: From now on, the panels are very similar to the ones shown in 8.4.1, “Installing on Windows, Linux, and Linux on POWER” on page 184.

6. The next panel displays the license information. Read and accept it. Click **Next**.
7. On the next panel, enter the phrase for secure key generation and click **Next**.

If you decide to skip it at this stage, you can manually generate a secure key after installation. Use the following two commands from QShell command line:

```
cd /qibm/proddata/iia  
IIA_TaskInvocation -task createKey -phrase <secure phrase>
```

Replace <secure phrase> with your phrase.

8. The next panel shows the installation pathname of IBM Installation Agent and total file size (see Figure 8-8). Click **Next** to begin the remote installation of the IBM Installation Agent on OS/400.



Figure 8-8 Location and total size of installation of IBM Installation Agent on OS/400

9. After installation completes, some summary information is displayed with a message about a successful installation (see Figure 8-9). Click **Finish** to exit from the InstallShield Wizard.

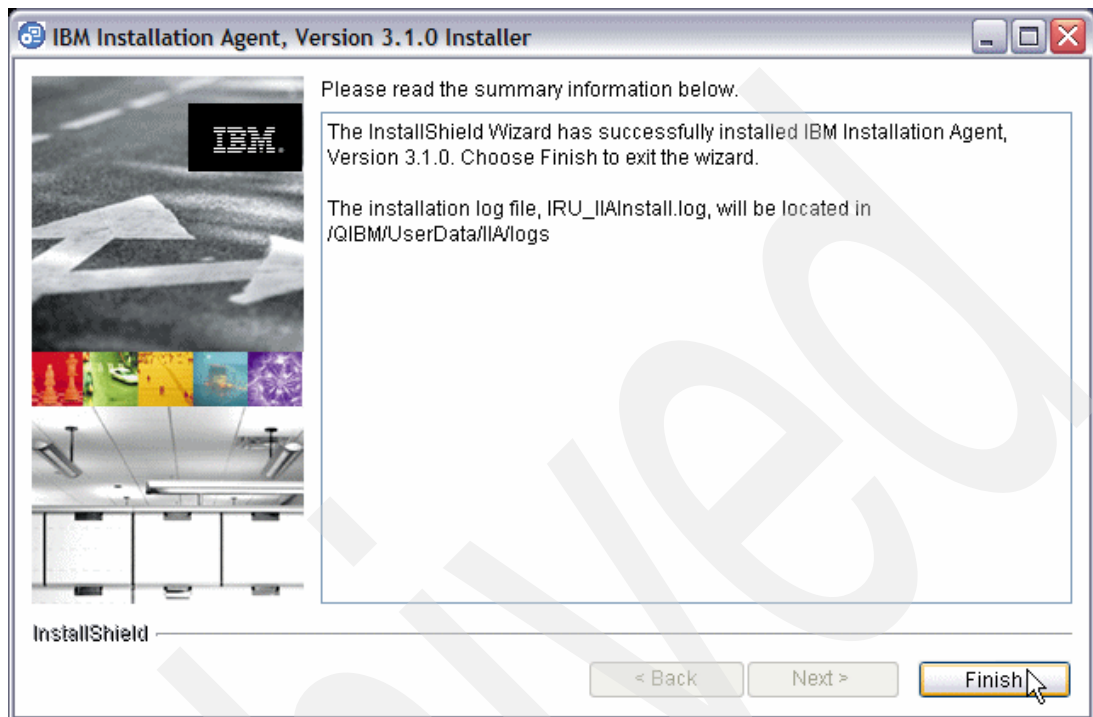


Figure 8-9 Summary information of Install Shield Wizard on OS/400

Validating installation on OS/400

You can verify if the installation of IBM Installation Agent worked fine with the OS/400 command:

```
WRKLNK 'qibm/proddata/iaa'
```

As shown in Figure 8-10, you can see files and directories that are created in IFS.

```
Work with Object Links

Directory . . . . : /qibm/proddata/iaa

Type options, press Enter.
  2=Edit  3=Copy  4=Remove  5=Display  7=Rename  8=Display attributes
  11=Change current directory ...

Opt  Object link      Type      Attribute  Text
-----
      _uninst          DIR
      externalSupportJar > DIR
      info             DIR
      license          DIR
      DJT_ibmnsit.jar   STMF
      DJT_IIA.properties STMF
      DJT_IIAsetup.iss  STMF
      IIA_Overview     DIR
      IIA_TaskInvocation STMF

Parameters or command
====> wrklnk '/qibm/proddata/iaa'
F3=Exit  F4=Prompt  F5=Refresh  F9=Retrieve  F12=Cancel  F17=Position to
F22=Display entire field  F23=More options

More...
```

Figure 8-10 wrklnk command on OS/400 system

Starting IBM Installation Agent on OS/400

For starting the IBM Installation Agent on an OS/400 system, you can do the following steps:

1. On the OS/400 command line, type the following command and press Enter:

```
qsh
```

2. Change to the IIA directory:

```
cd /qibm/proddata/IIA
```

3. Run the following script:

```
IRU_iaa_start-agent
```

4. The following message should be displayed (see Figure 8-11):

```
CPC1221: Job 052962/ITSCID04/QIIASRV submitted to job queue QSYSNOMAX in library QSYS
```

```

QSH Command Entry

> ls
DJT_IIA.properties      IRU.properties          _uninst
DJT_IIAsetup.iss        IRU_iaa_start-agent     externalSupportJars
DJT_ibmnsit.jar         IRU_iaa_stop-agent      info
IIA_Overview            IRU_iaa_uninstall-agent license
IIA_TaskInvocation      IRU_iaa_version
$
> pwd
/qibm/proddata/iaa
$
> ./IRU_iaa_start-agent
CPC1221: Job 054117/ITSCID04/QIIASRV submitted to job queue QSYSNOMAX in lib
rary QSYS.
$

===>

F3=Exit   F6=Print F9=Retrieve F12=Disconnect
F13=Clear F17=Top  F18=Bottom F21=CL command entry

```

Figure 8-11 Output of start IBM Installation Agent on OS/400 system

5. To check if the IBM Installation Agent is active, you can use the WRKACTJOB command and check if the QIIASRV job is running as shown in Figure 8-12.

```

Work with Active Jobs                                RCHASRAL
                                                    02/02/05 11:38:13
CPU %:      .2      Elapsed time: 00:00:46      Active jobs: 179

Type options, press Enter.
  2=Change  3=Hold  4=End  5=Work with  6=Release  7=Display message
  8=Work with spooled files 13=Disconnect ...

Opt  Subsystem/Job  User      Type  CPU %  Function      Status
-----
   Q CSTCTHRMD    QSYS      BCI    .0    PGM-QCSTCTEXEC SELW
   Q CSTCTRMCD    QSYS      BCI    .0    PGM-QCSTCTEXEC SELW
   Q CSTSRC D    QSYS      ASJ    .0    PGM-QCSTCTSRC D SELW
   Q DIRSRV      QDIRSRV   BCH    .0    PGM-QGLDSVR    SIGW
   Q GLDPUBA     QDIRSRV   ASJ    .0    PGM-QGLDPUBA   SIGW
   Q GLDPUBE     QDIRSRV   ASJ    .0    PGM-QGLDPUBE   DEQW
   Q HTTP       QTMHHTTP  BCH    .0    PGM-QPMPUDCDRV SELW
   Q IASRV       ITSCID04  BCH    .0    CMD-STRQSH     TIMW
   Q IJSSCD     QIJS      BCH    .0    PGM-QIJSCMON   DEQW

More...

Parameters or command
===>
F3=Exit   F5=Refresh   F7=Find   F10=Restart statistics
F11=Display elapsed data F12=Cancel F23=More options F24=More keys

```

Figure 8-12 Display active job on OS/400 system

8.5 Deployment scenarios

In this section we describe the step-by-step deployment process for three scenarios:

- ▶ Local deployment from a development system (staging server)
- ▶ Local deployment from CDs
- ▶ Remote deployment from a development system

In our tests, we used the development system as a staging server. After generating the deployment code for the sample solution (Trade6 and Order Entry), the development system contains all necessary software to deploy a solution.

If you are considering creating a staging server on a separate system, you need to install Deployment Wizard. One of the Express Runtime V2.1 installation options is *Deployment Wizard Only installation* (see Figure 8-13). For information, see “Deployment Wizard Only Installation” on page 46.



Figure 8-13 Installing Deployment Wizard only

To support remote deployment on a dedicated staging server (a different system from your development machine), you need to do the following actions:

- ▶ Export the solution files to the Staging Server (see 7.1, “Creating the solution package for Trade6” on page 166).
- ▶ Install the IBM Installation Agent (IIA) on all Target Systems. It allows network communication between the Staging Server and the Target Machine. During installation on IIA, you can also generate a secure key.

Then you start IIA. It opens a TCP Listener on a specific port. At this time, a staging server can connect to a target system.

- ▶ If you haven’t specified a secure phrase during IIA installation, you need to do it manually. You have to restart IIA after creating or modifying a secure phrase.

Secure Key: You must specify the same secure phrase on the staging server and each target computer in order for the authentication to be successful. The secure phrase can be any combination of alphanumeric characters and can be any length. Make the phrase something you can remember, but not something that is easy for someone else to guess. If a staging server makes five consecutive attempts to deploy to a target computer without correctly matching security keys, the security keys on the target computer will be deleted. You must then go to the target computer and create the security keys again using the Key Manager utility.

The Key Manager utility protects against unauthorized use.

The third way is to deploy using media. The last scenario that we can imagine is that from the Development System, we generate a set of CDs. We can use it to deploy on the Target System. It means that installing the solution using CDs which contains setup programs and files needed. To do that we can use Solution Launcher.

We remind you about this topic in Chapter 7, “Packaging a solution” on page 165.

In any other cases of deployment, both locally and remotely, we use the Deployment wizard tool as well.

8.5.1 Scenario 1: Deploying to a local system

We describe step-by-step deployment of the Trade6 application on a local Windows system. Remember that we have both development and deployment environments on this system (refer to Chapter 6, “Developing a wrapper” on page 91 for development tasks):

1. Click **Start** → **Programs** → **IBM Express Runtime 2.1** → **Deployment Wizard**.
2. Click **File** → **Open**. By default, Deployment Wizard opens in the directory, C:\Program Files\IBM\Runtime21\SolutionEnabler.
3. Locate the Trade6 solution (.ser) file (see Figure 8-14).

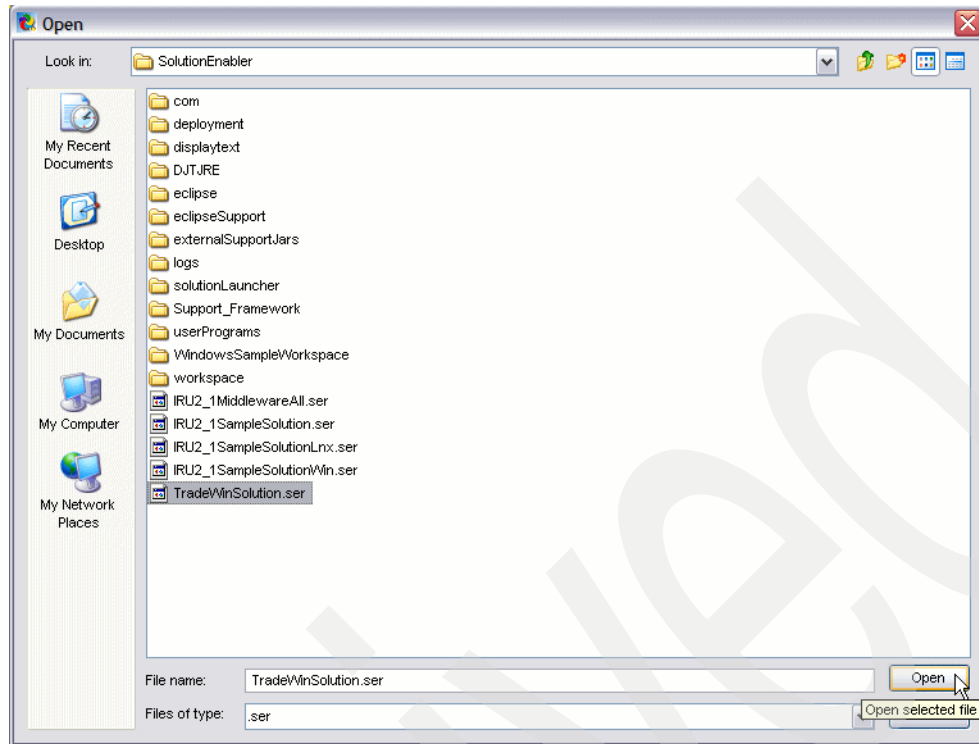


Figure 8-14 Deployment Wizard - Choice solution file

4. Deployment Wizard loads the TradeWinSolution.ser file. Click **Next** to begin the deployment process (see Figure 8-15). This process installs all middleware and Trade6.

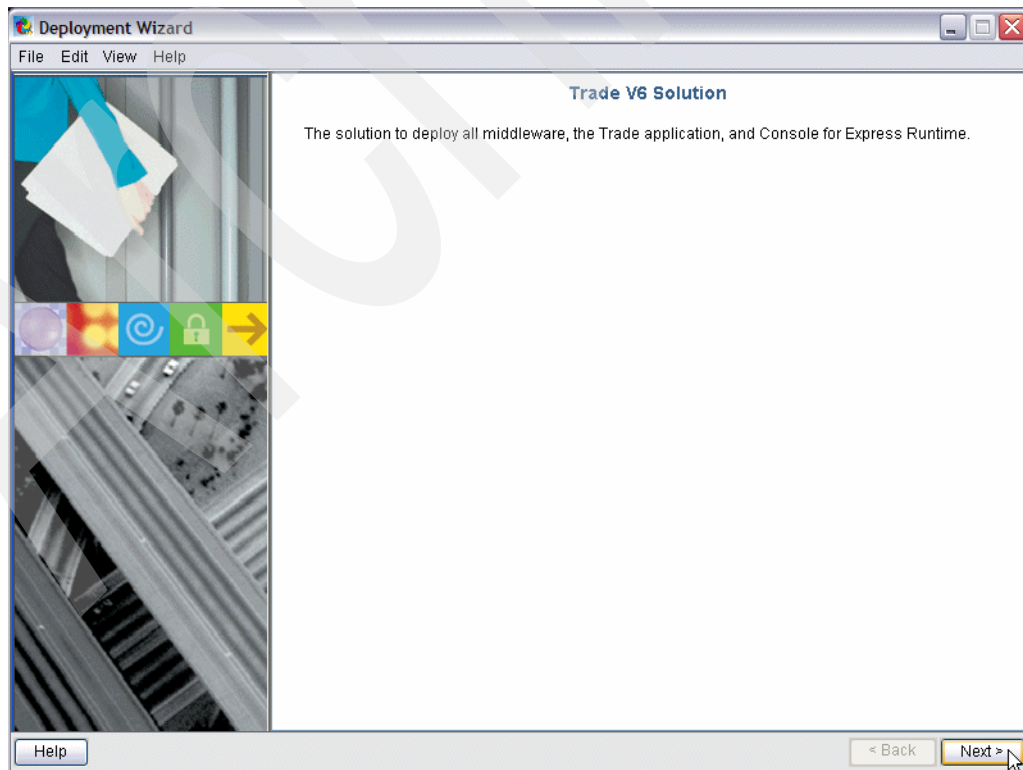


Figure 8-15 Deploying Trade Win Solution

5. TradeWinSolution.ser includes only one task. In this panel, select the **Trade application** checkbox and click **Next** as shown in Figure 8-16.

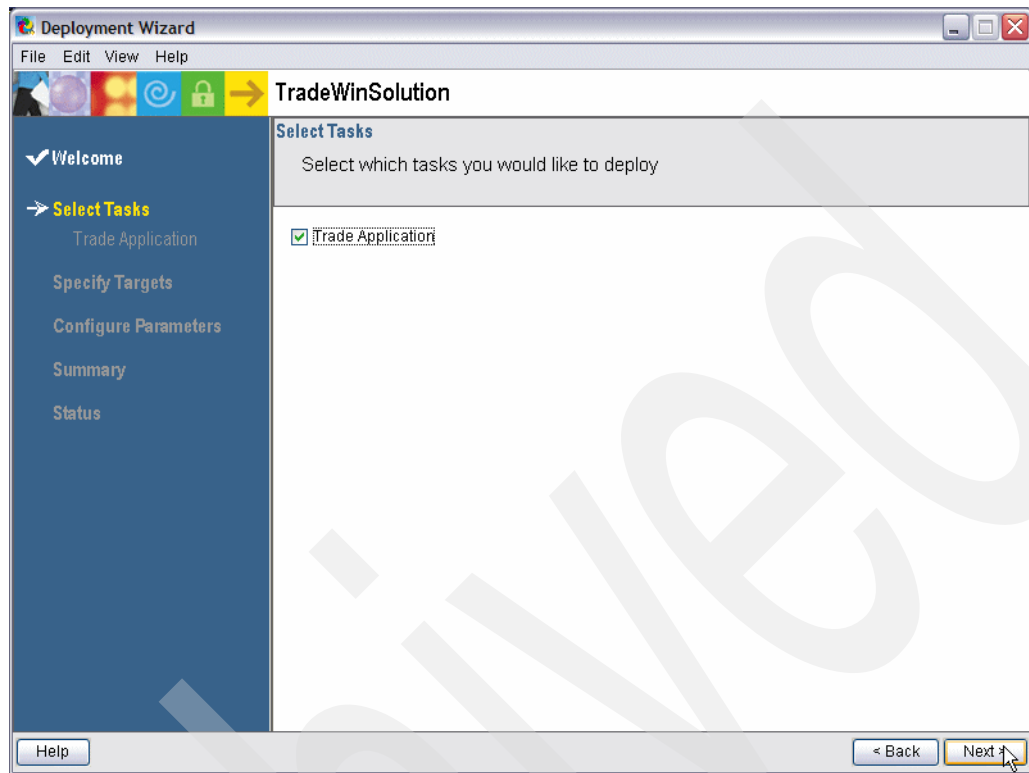


Figure 8-16 Selecting tasks

6. Now select the platform where the middleware and Trade application will be installed. This example is built for Windows as described in Chapter 6, “Developing a wrapper” on page 91. It has only one application. Select **Trade Application and Middleware for Windows** and click **Next** (see Figure 8-17).

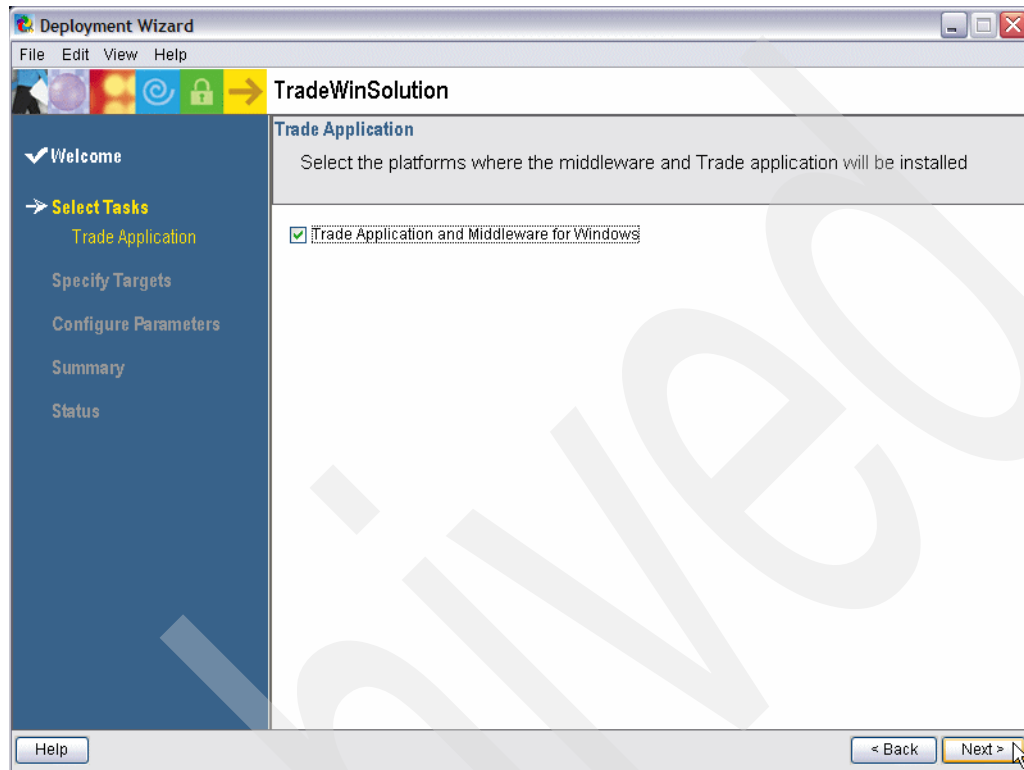


Figure 8-17 Selecting platforms for deployment

7. Now specify the Target computer for the Middleware and Trade Application:

- Type **localhost** in target computer field and click **Add**.
- Click the **Test connections** button for testing the connection.

A windows popup is displayed with the results of the test as shown in Figure 8-18.

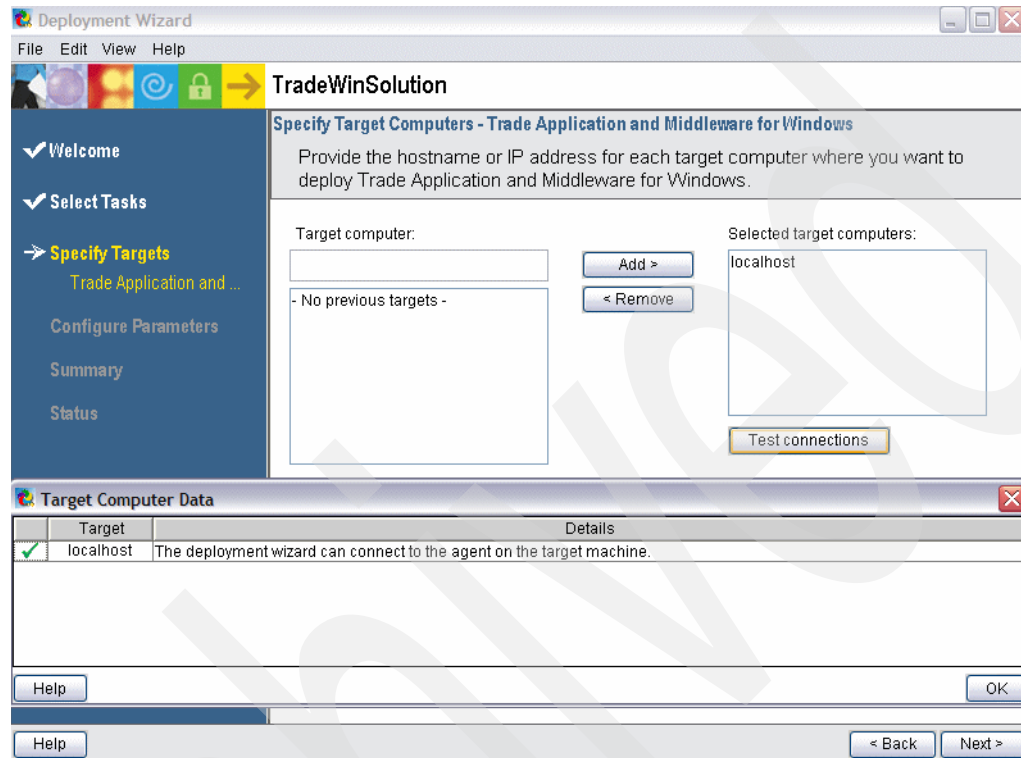


Figure 8-18 Testing connection on localhost

8. Now you must provide configuration parameters for your solution. The first panel is for DB2 installation. The target directory and some other fields are filled with the default values, but you can change them. However, you have to specify two parameters:

- DB2 Administrator Password
- Verify password

Click **Next** to continue as shown in Figure 8-19.

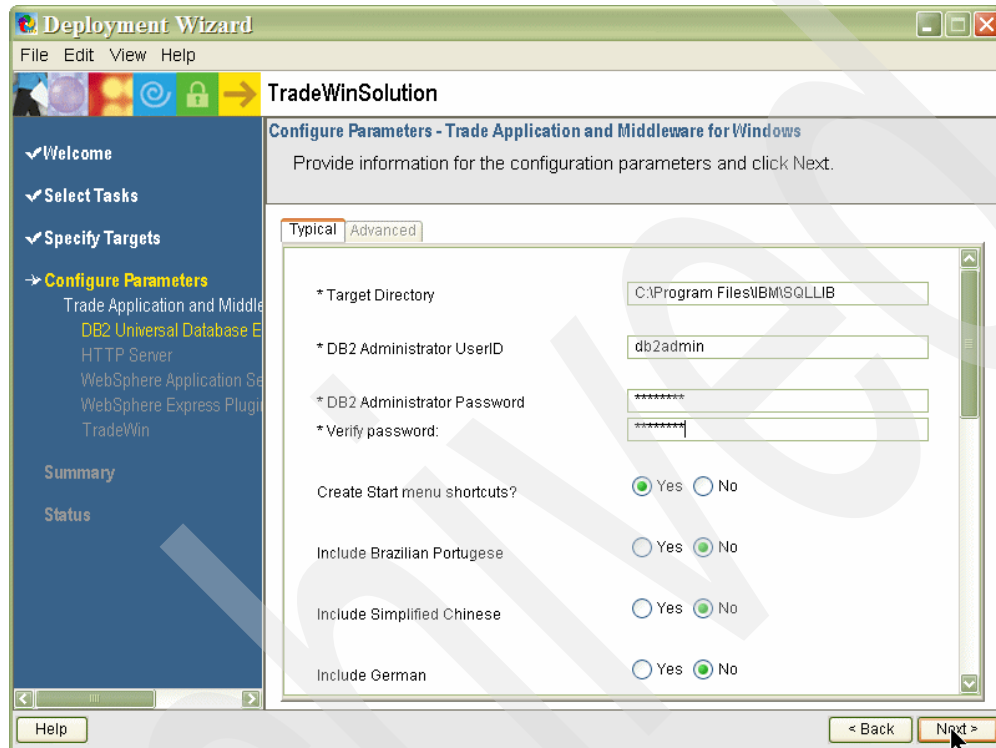


Figure 8-19 Configure parameters for DB2

9. The next panel shows the parameters for the IBM HTTP Server 6.0 installation. Like other middleware, the installation directory and TCP ports are filled in with the default values. Check if they are correct on your system and click **Next** to continue (see Figure 8-20).

Note: If a middleware component is already installed, Deployment Wizard skips installation of this component. A radio button is present in each middleware component panel that enables product re-installation as shown in Figure 8-20. For more details about what to do if a component is already installed, see “What if you have one component installed?” on page 224.

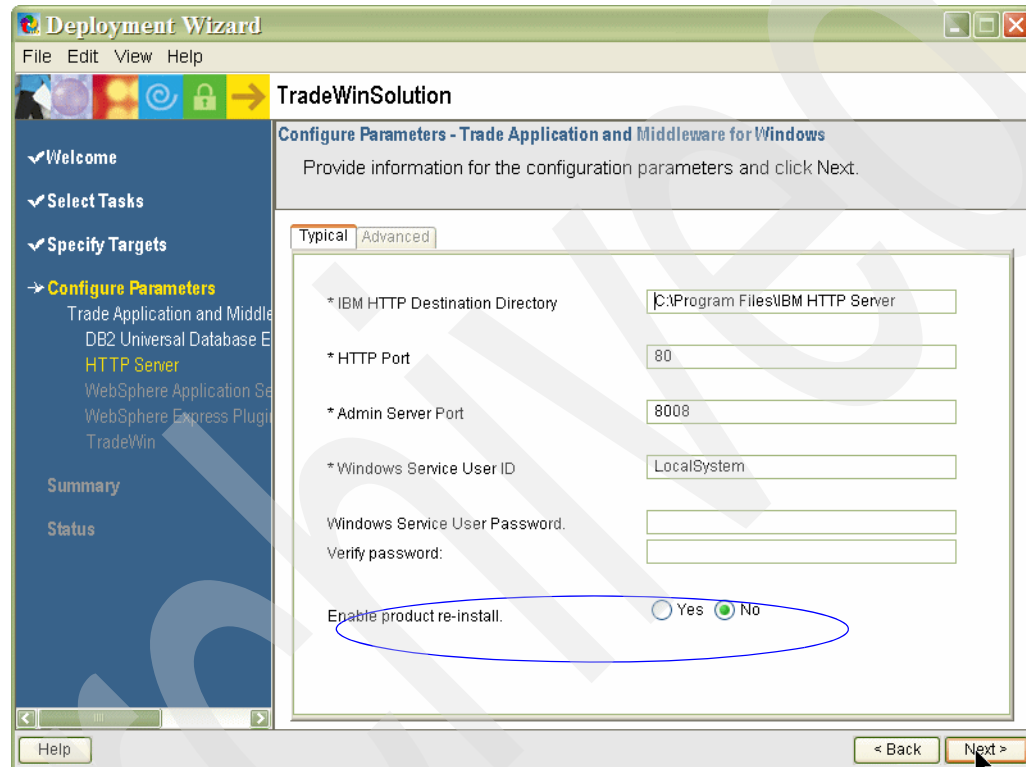


Figure 8-20 The HTTP server panel

10. Now you configure parameters for IBM WebSphere® Application Server - Express Version 6. The installation directory and TCP ports are filled with the default values:

- Check for target directory
- Check if TCP ports are available on your computer

Click **Next** to continue (see Figure 8-21).

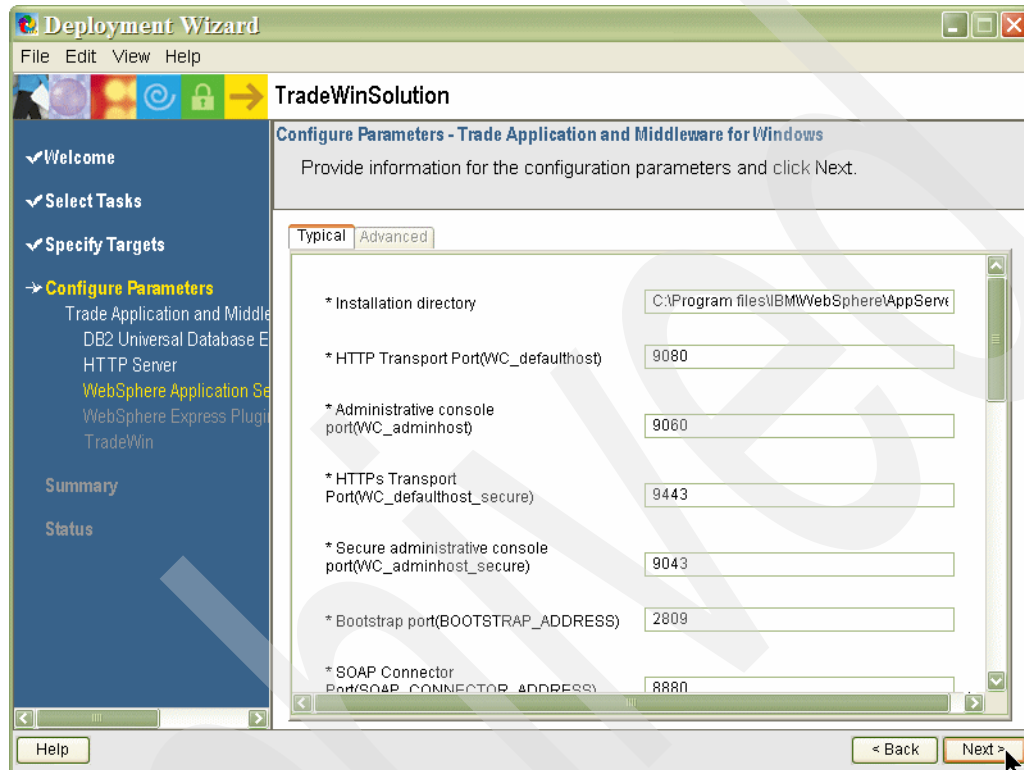


Figure 8-21 WebSphere installation parameters

11. The next panel is for IBM WebSphere® Application Server - Express Version 6 Plug-in for IBM HTTP Server 6.0 installation. You must provide the hostname **only** if it is a remote deployment. In this example, we deploy the solution locally. The other fields are filled in with the default values. Click **Next** to continue (see Figure 8-22).

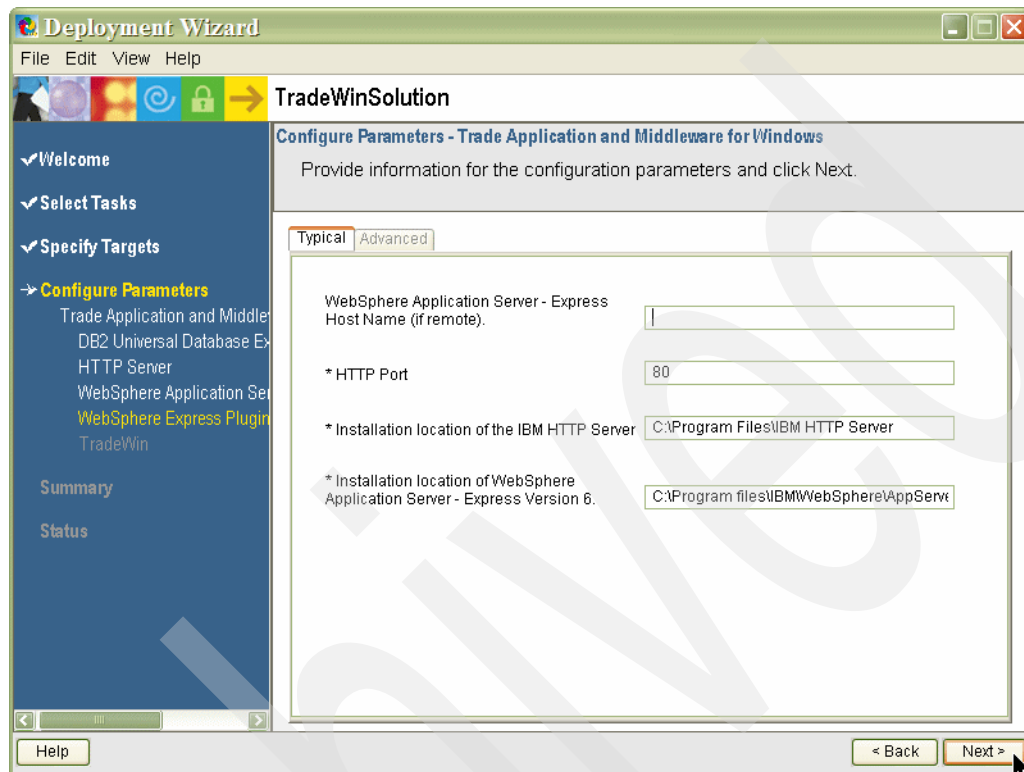


Figure 8-22 Web server plug-in panel

12. The next panel presents the Trade6 parameters. Configure the following DB2 parameters for the Trade application: DB2 Administrator User ID and password (see Figure 8-23). These fields contain the values that we have entered on the DB2 panel. If you plan to use a different DB2 user ID to access the database from the Trade6 application, replace these values.

Important: This is a good example to show two important points:

- ▶ Application variables
- ▶ Sharing a variable

The DB2 administrator ID and password are two variables that we defined in step c on page 103. Notice that the DB2 administrator ID variable has the default value and we can leave it as is (see Figure 8-23). However, we didn't provide a default value for the password.

The reason why you see the password fields filled is because we have shared these two variables between all applications in the solution (see step 15 on page 141 through step 22 on page 142). So now, once we set these values in one place, their values are available to all applications in the solution.

Click **Next**.

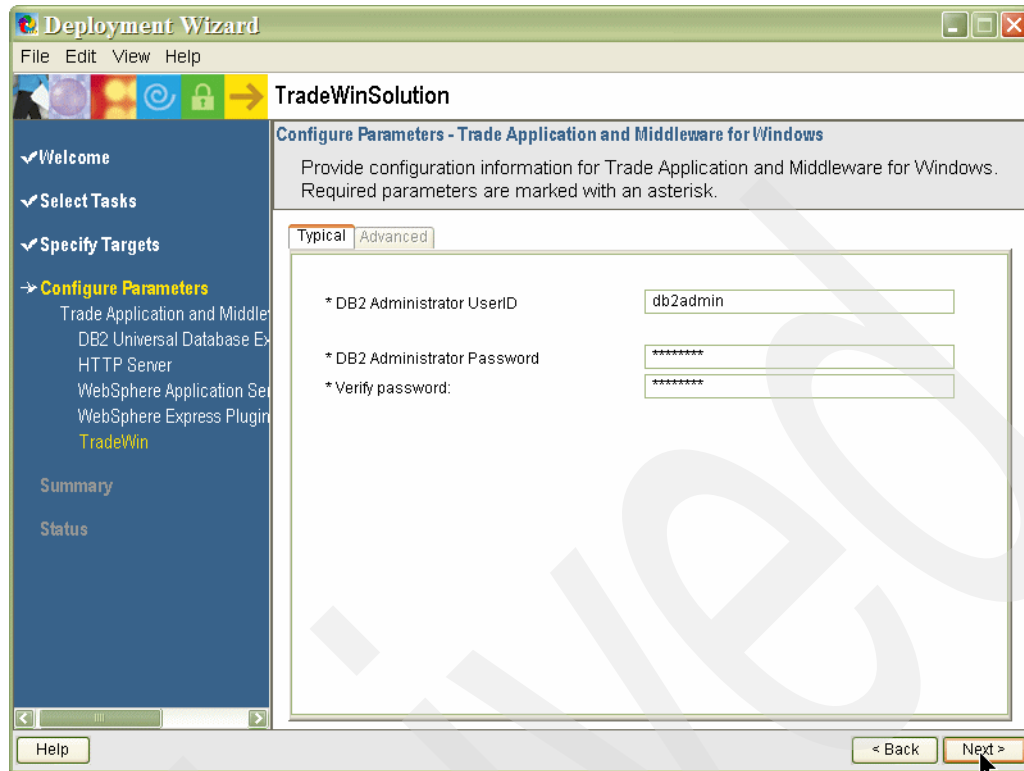


Figure 8-23 Trade application requirement on deployment wizard.

13. The Summary panel is shown next. In this example you have only one task, so you can either click **Deploy all** or **Deploy Task**. An estimated deployment time *f* is displayed.

Note: Installation time will be less if some of the middleware components are already installed.

14. Deployment start. The status bar shows the progress.

Note: If Deployment Wizard can't find all the required solution files, you may see a pop-up window asking for the location of those files. Figure 8-24 demonstrates this point. In this example we can click **Browse** and navigate to the Trade_ScriptsDir directory.

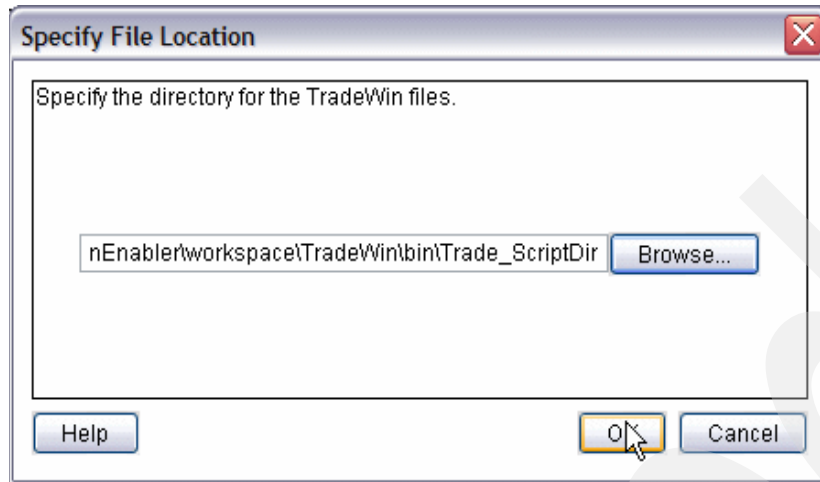


Figure 8-24 Specify application files location of Trade6

15. When installation completes, a message of successful deployment is displayed as shown in Figure 8-25. You can click **Master log** to open the IRU_DeploymentWizard.log file or click **Close** to finish the solution deployment.

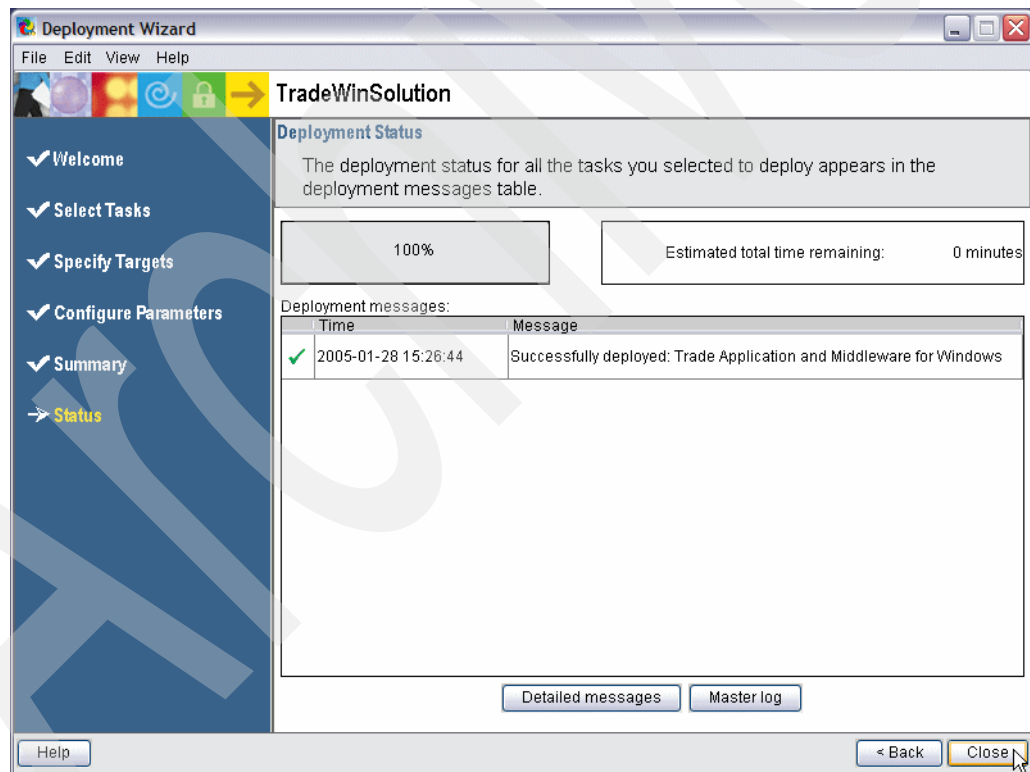


Figure 8-25 Successful deployment.

8.5.2 Scenario 2: Deploying from CDs

When you have a set of CDs or DVDs with your solution, you are ready to deploy this solution either locally or remotely. For this scenario, we deploy the Trade6 solution for Windows.

You need to perform the following steps to deploy that solution (using a set of CDs in this example):

1. Insert CD1 into a CD-ROM driver of your workstation.
2. If autorun is enabled, Solution Launcher starts automatically. If it doesn't enabled, then you need to double-click the IRU_WindowsSetup.exe file on CD1.
3. Solution Launcher is loaded. After some time you should see a pop-up window where you need to select a language for the installation. Select your language and click **OK**.
4. The Deployment Wizard installation starts. Click **Next** on the first panel (see Figure 8-26).

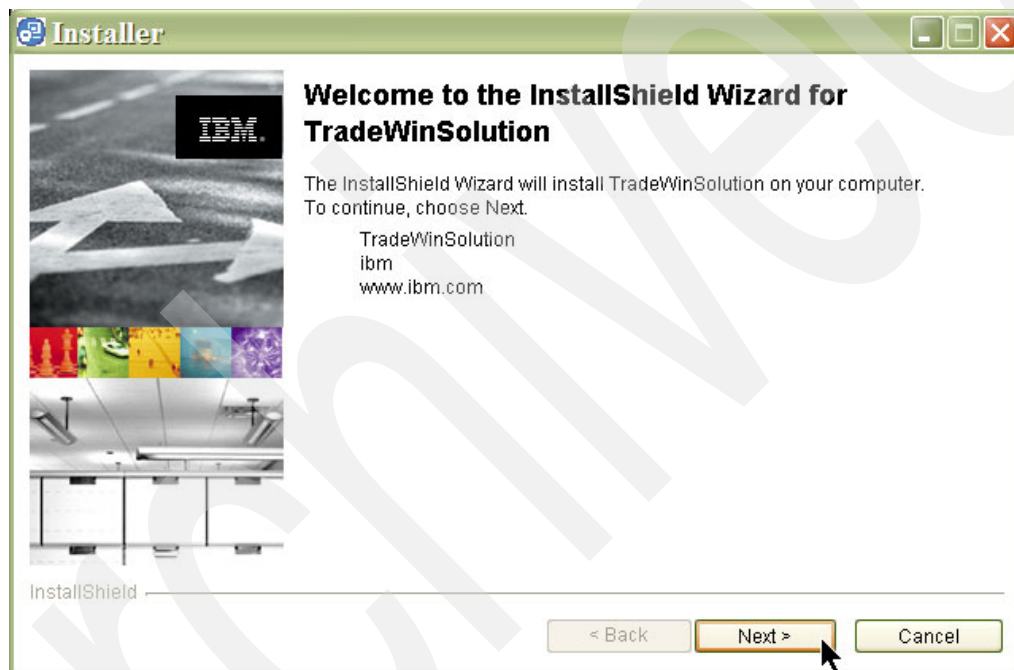


Figure 8-26 Welcome page

5. On the next panel, type the directory name for the temporary files. If the directory doesn't exist, it will be created. Click **Next**.
6. The next panel shows the summary of the selected options. Click **Next**.
7. The installation starts.
8. If the solution includes more than one CD/DVD, Deployment Wizard will display a pop-up window prompting for the location of the components. Place the second CD/DVD, type the drive letter for your CD-ROM or DVD-ROM, and click **OK**. For example:

D:\

Note: Deployment Wizard allows you to use the Browse button to select the drive. However, there is a bug in Deployment Wizard. So, instead of *selecting* a drive, you have to *type* it.

9. Repeat the previous steps for other CDs/DVDs.

10. You should see the status bar as shown in Figure 8-27.



Figure 8-27 Installing the required files

11. When the installation completes, click **Finish**.

12. The Deployment Wizard starts (see Figure 8-28).

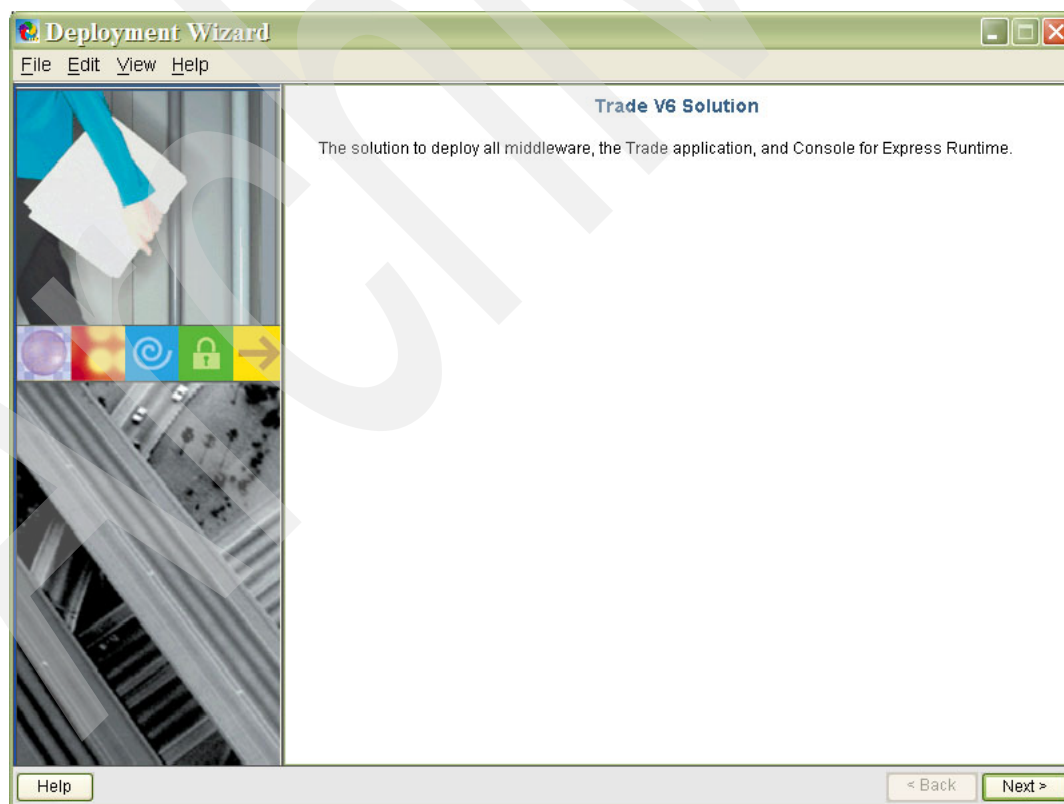


Figure 8-28 The first panel of Deployment Wizard

13. From this point on you follow the steps outlined in the previous scenarios:

- **For local deployment:** From step 4 on page 197 (in 8.5.1, “Scenario 1: Deploying to a local system” on page 196)
- **For remote deployment:** From step 4 on page 211 (in 8.5.3, “Scenario 3: Deploying to the remote systems (two systems)” on page 209)

8.5.3 Scenario 3: Deploying to the remote systems (two systems)

In this example we deploy the Trade6 solution (see Chapter 6, “Developing a wrapper” on page 91) to two remote systems:

- ▶ IBM HTTP Server 6.0 to a Windows system — Target System A
- ▶ IBM WebSphere® Application Server - Express Version 6, Trade6 J2EE application, and DB2 UDB Express 8.2 to another Windows box — Target System B

Preparing a solution for deployment on two or more systems requires some additional thought. You need to edit your solution’s `solution.xml` file in such a way that it reflects your deployment pattern.

For our example, we need to have two installation tasks:

- ▶ One task to install IBM HTTP server
- ▶ A second task to install the rest of the middleware and Trade6 application

To match these requirements, the Tasks tab of the `solution.xml` file should look as shown in Figure 8-29.

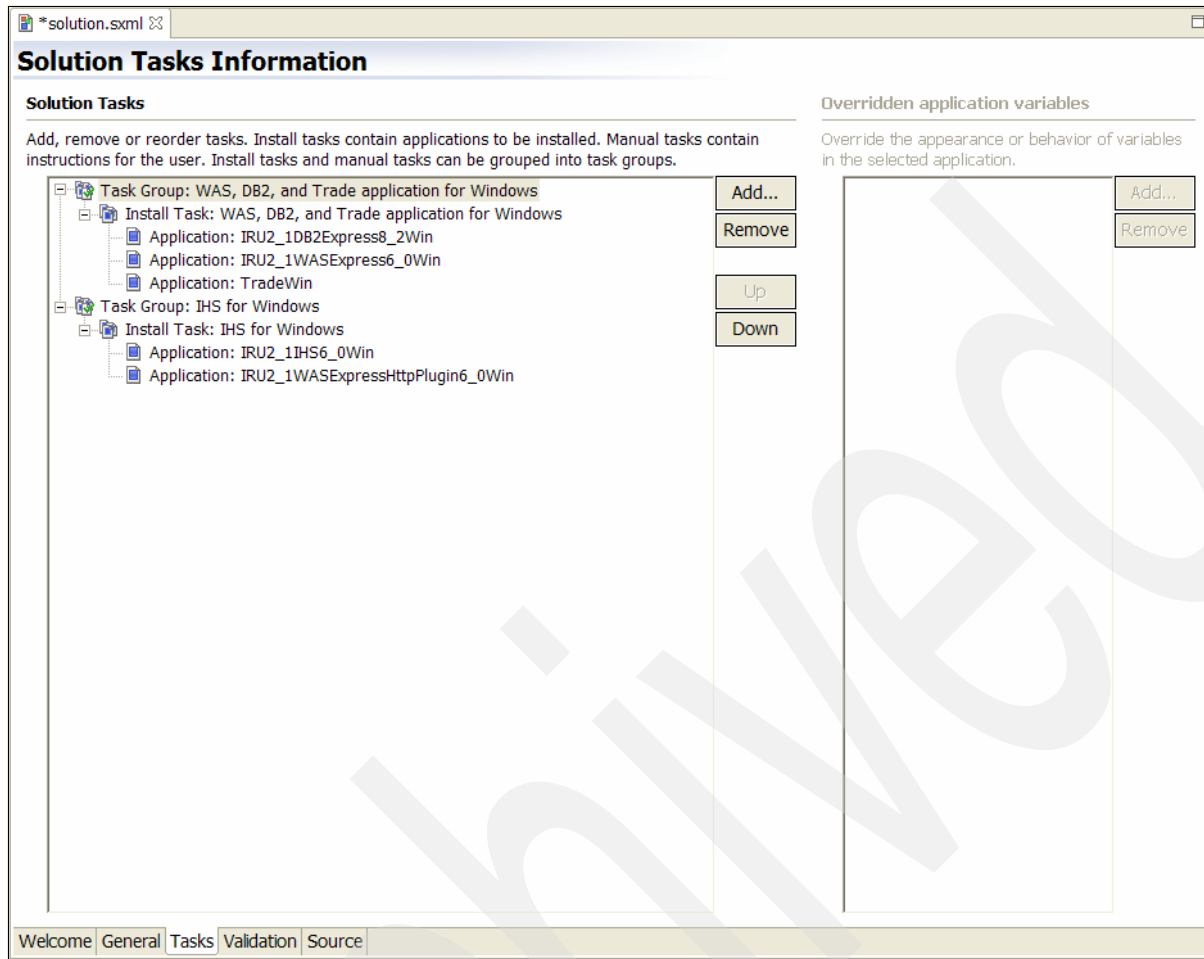


Figure 8-29 Preparing a solution for deployment in a distributed environment

After this modification, we re-generated the deployment packages for the TradeWin application and TradeWinSolution solution (see 6.2.6, “Building the Trade6 solution” on page 143).

The next step in preparation for deployment is to set up a Staging Server. It requires:

- ▶ Installation of Deployment Wizard. See “Deployment Wizard Only Installation” on page 46.
- ▶ Copying the following solution files from the development system to the Staging Server.

TradeWinSolution.ser.

This is a solution serialized file and it is created by the Development System when you export Trade6 solution. This file is located in C:\Program Files\IBM\Runtime21\SolutionEnabler directory on Development System as we had described in 7.1, “Creating the solution package for Trade6” on page 166. For remote deployment, you have to copy this file from the Development System to the Staging Server anywhere in the file system. When Deployment Wizard starts and you click **File** → **Open**, you can specify the directory where you placed TradeWinSolution.ser. The directory, C:\Program Files\IBM\Runtime21\SolutionEnabler, is the **default pathname**.

tradewin.xx.userPrograms.jar.

This is the user program's jar file and it is created by Development System when you export the Trade6 solution. This file is located in the directory, C:\Program Files\IBM\Runtime21\SolutionEnabler\userPrograms on the Development System. You have to copy this file into the same directory on the Staging Server. If the Deployment Wizard doesn't find it at this location, the deployment fails.

tradewin.xx.jar.

This is the deployment package .jar file and it is created on the Development System when you generate a deployment package for TradeWin application. C:\Program Files\IBM\Runtime21\SolutionEnabler\workspace\IRU_common_resources\mediaJars is the default pathname on Development System. You can copy this file anywhere in the Staging Server's file system. However, you may need to point to this file during the deployment (see Figure 8-24 on page 206). To skip this additional step of pointing to the tradewin.xx.jar file, you should copy it to C:\Program Files\IBM\Runtime21\SolutionEnabler.

Now that we have done all of the required preparation steps, let us deploy the solution:

1. On two Target Computers you have to start IBM Installation Agent now. Perform these steps on each system:
 - a. Click **Start** → **Program** → **IBM Installation Agent 3.1** → **First Steps**
 - b. Click **Start the Agent**

Note: The IBM Installation Agent on the Target Computers must be started before performing remote deployment.

2. On the staging server, click **Start** → **Programs** → **IBM Express Runtime 2.1** → **Deployment Wizard**.
3. Click **File** → **Open**. By default, Deployment Wizard opens the C:\Program Files\IBM\Runtime21\SolutionEnabler directory. Select **TradeWinSolution.ser** as shown in Figure 8-14 on page 197.
4. Deployment Wizard loads the TradeWinSolution.ser file. Click **Next** to begin the deployment steps.

5. TradeWinSolution.ser includes two task groups. The first one includes the Trade6 application with DB2 UDB Express 8.2 and IBM WebSphere® Application Server - Express Version 6. The second is for the IBM HTTP Server 6.0 and WebSphere Plug-in.

Click both **Trade Application** and **IHS for Windows** checkboxes and then click **Next** as shown in Figure 8-30.

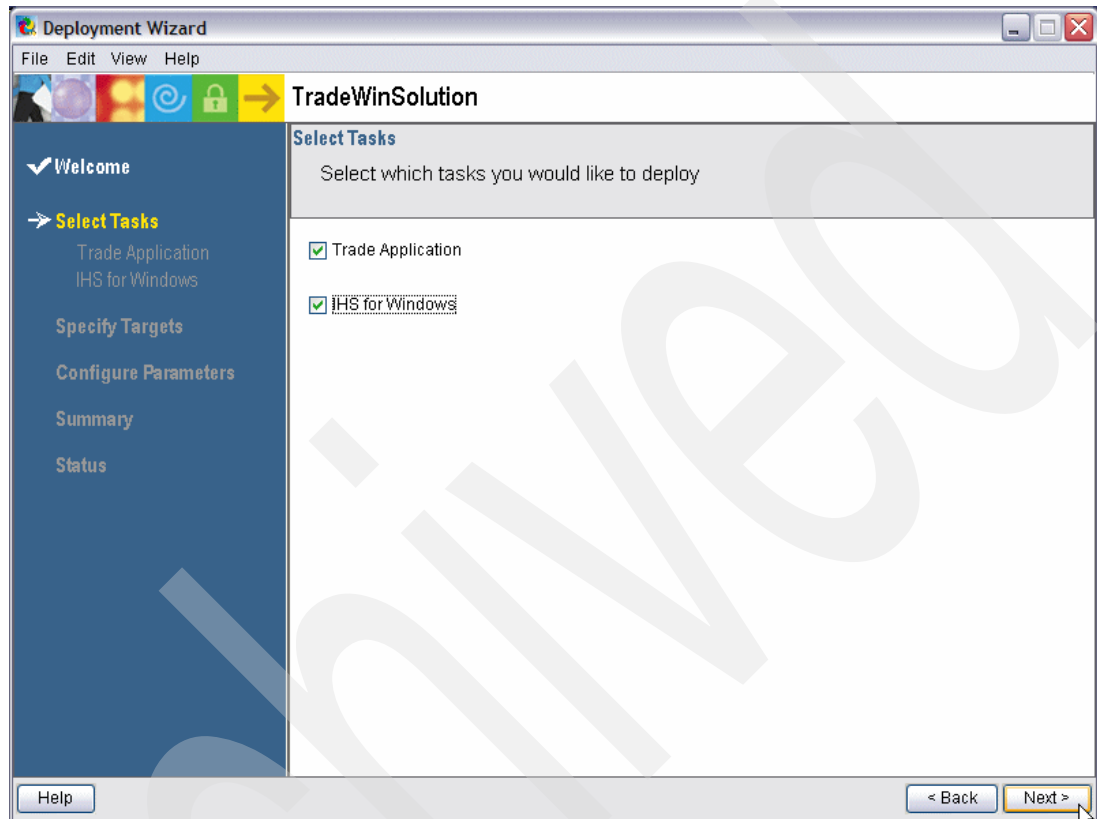


Figure 8-30 scenario 2: Selecting tasks group

6. Select the first task that will be installed on the first Windows box as shown in Figure 8-31. Click the **WAS, DB2, and Trade Application on Windows** checkbox. Click **Next**.

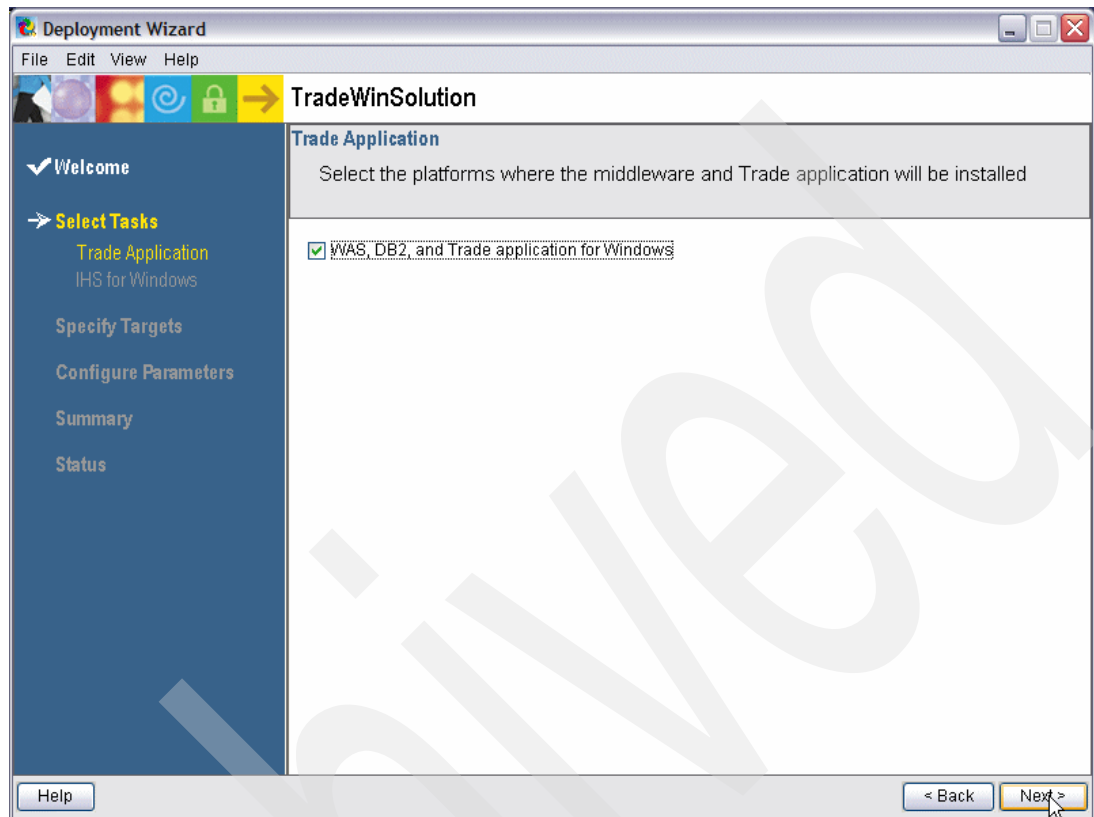


Figure 8-31 Scenario 2: Selecting task for WebSphere and DB2

7. Select the second task that will be installed on the second Windows box as shown in Figure 8-32: Click the **IHS for Windows** checkbox. Click **Next**.

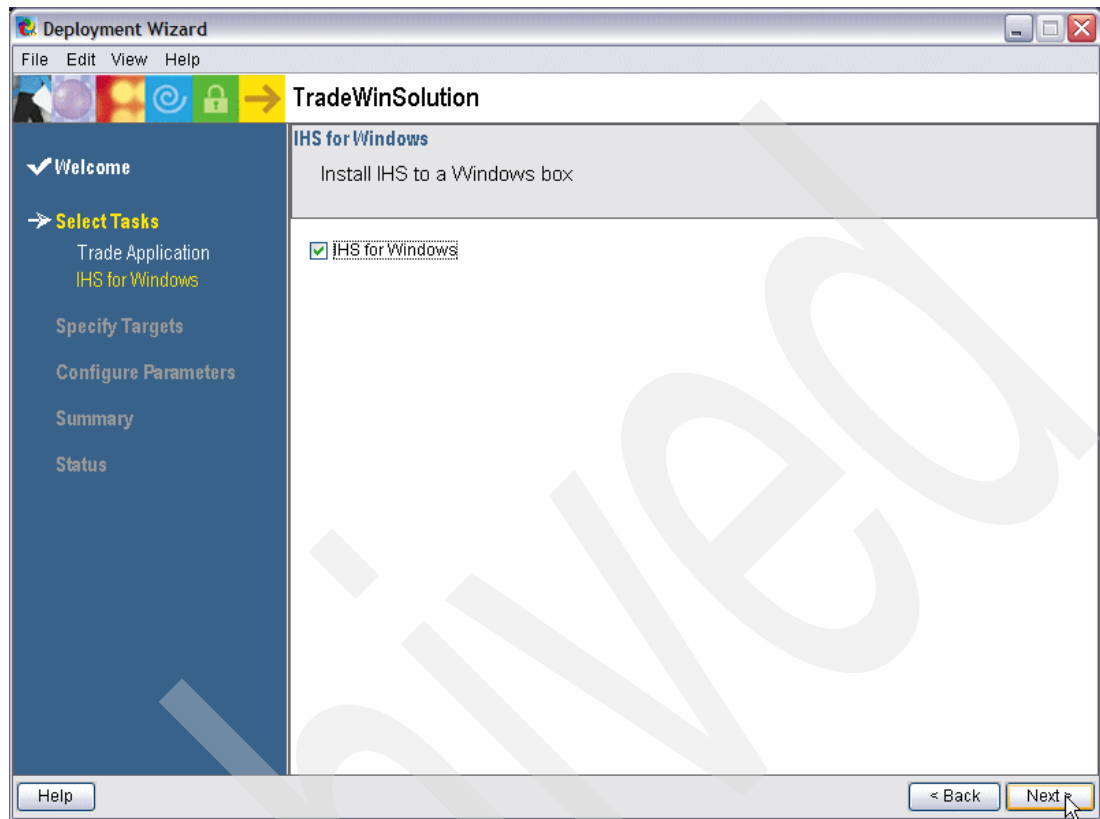


Figure 8-32 Scenario 2: Selecting task for IBM HTTP Server and WebSphere Plug-in

8. On the next panel, specify the Target computer with a fully qualified name. This is the target system where Trade5, WAS, and DB2 will be installed. Click the **Add** button.
9. Select this system in the Select target computers box and click **Test connections**.
10. A pop-up window (see Figure 8-33) is displayed showing the test results. Click **Next**.

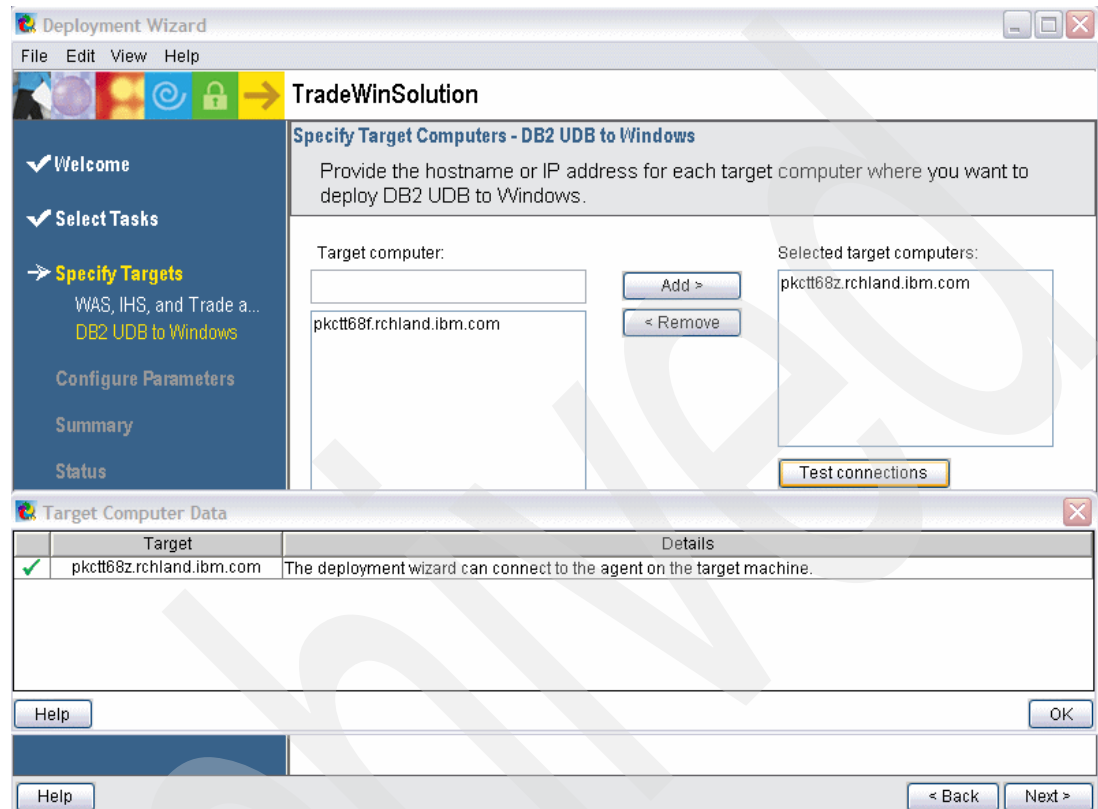


Figure 8-33 Scenario 2: Testing Connection for Target Computers

11. On the next panel, specify and test the second target system in the same way as you did for the first one. Click **Next**.
12. You must now provide the configuration parameters required for your solution. The first panel is for DB2 installation. The target directory and some other fields are filled with the default values (see Figure 8-34), but you can change them.

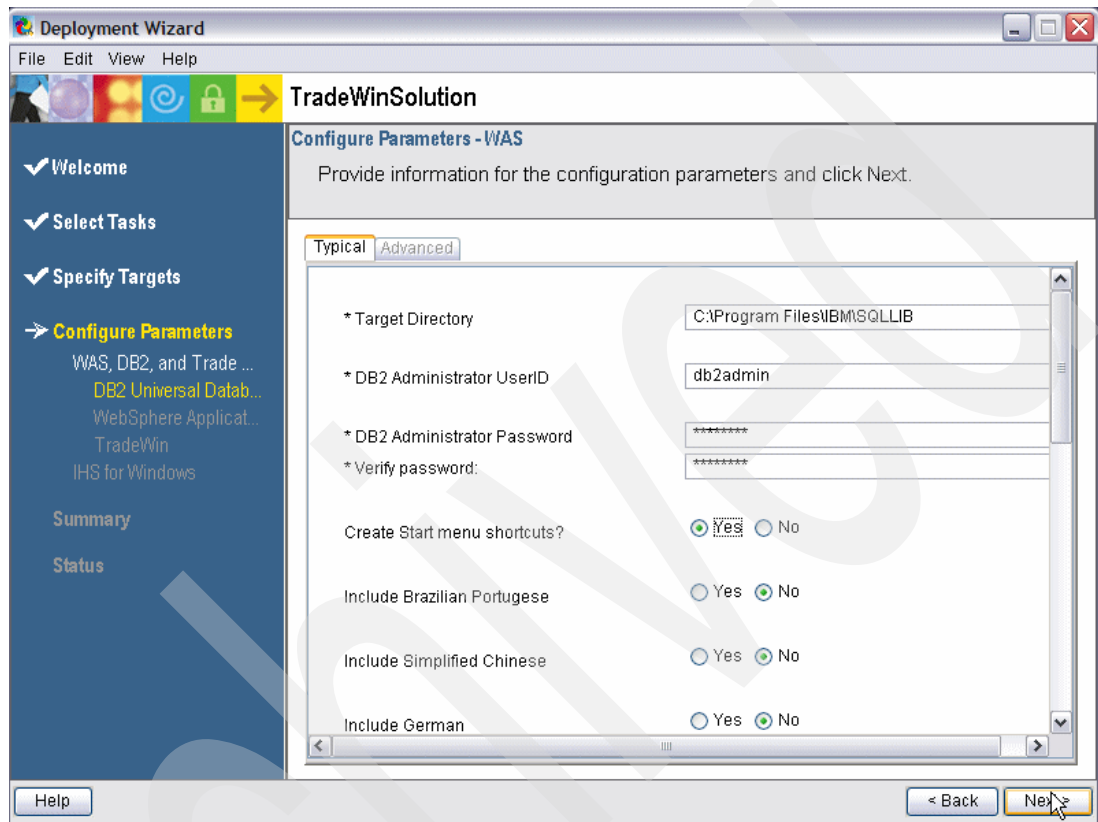


Figure 8-34 Scenario2: Configuring parameter for DB2

13. The next panel displays the parameters for the IBM WebSphere® Application Server - Express Version 6 installation. The target directory and TCP port fields are filled with the default values, but you can change them. Check if ports are available on your Windows Target Computer and click **Next** to continue as shown in Figure 8-35.

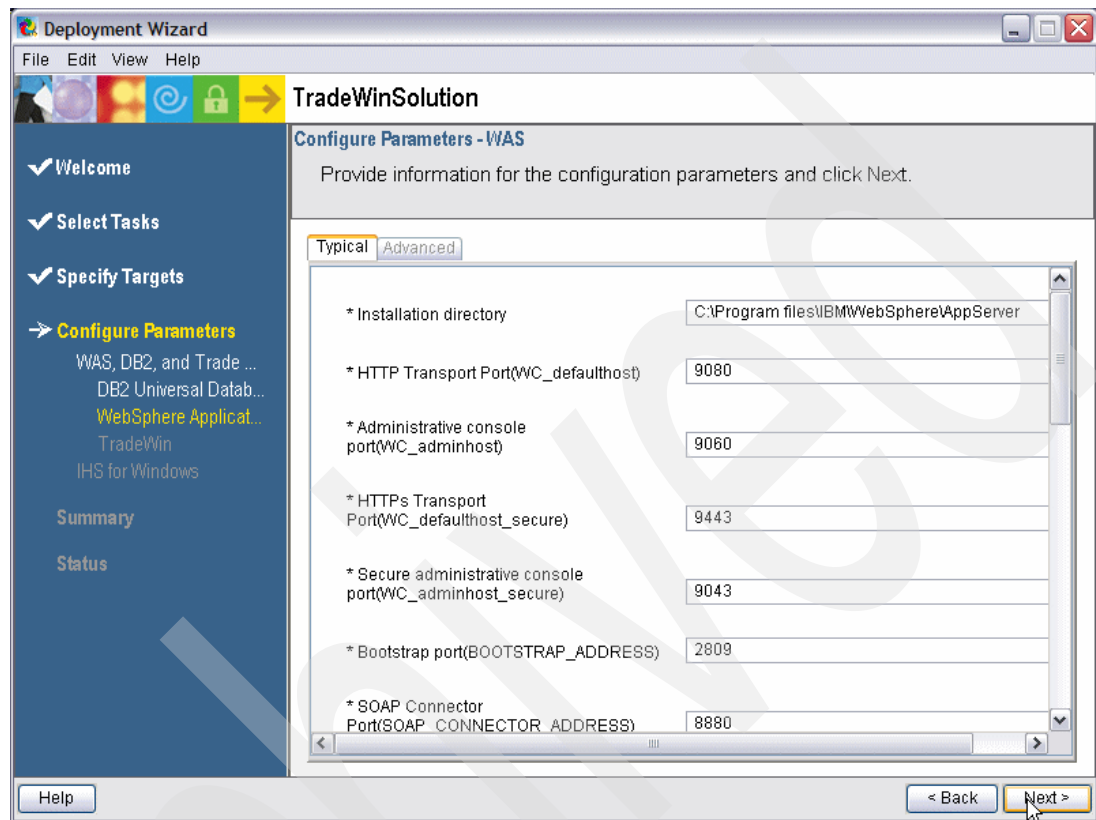


Figure 8-35 Example 2: Configuring parameter for WebSphere Application Server

14. The next panel shows the required parameters for the Trade6 application. In this example we need only the DB2 Administrator User ID and password. These fields should be filled in with the values that we entered in step 12 on page 216. Click **Next** as shown in Figure 8-36.

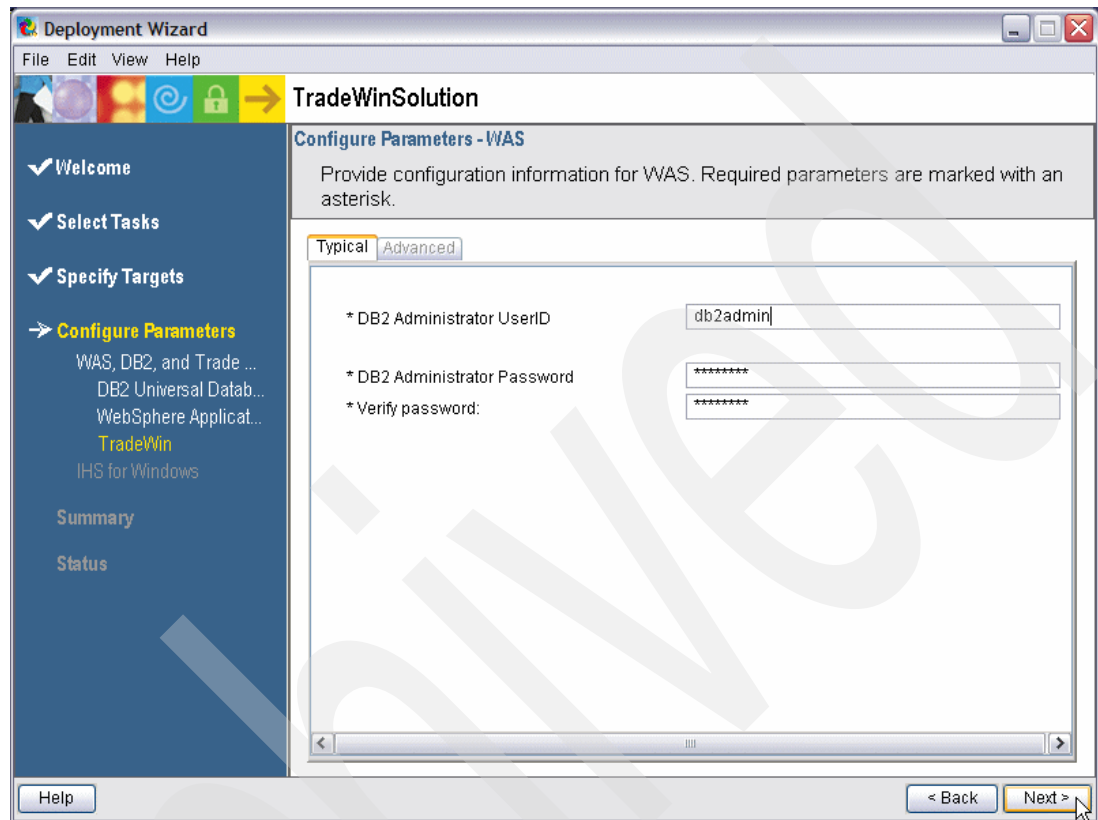


Figure 8-36 Example 2: Configuring parameter for Trade application

15. The next panel is to configure parameters of the second task group (see Figure 8-37). This second task group will be performed on the second Target Computer. It installs the IBM HTTP Server and the corresponding plug-in. The target directory and TCP ports fields are filled in with the default values, but you can change them. Click **Next** to continue.

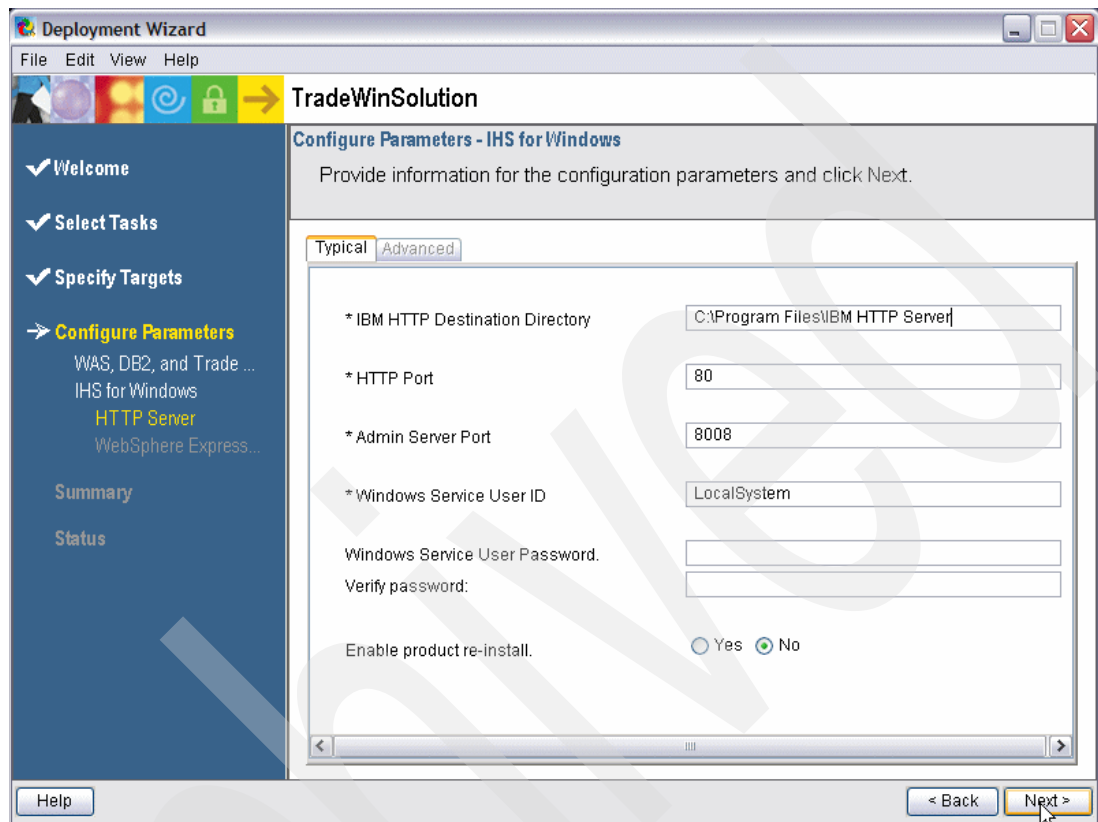


Figure 8-37 Example 2: Configuring parameter for IBM HTTP Server

16. The configuration parameter for the IBM WebSphere® Application Server - Express Version 6 Plug-in for IBM HTTP Server 6.0 is now required. In this example we want to deploy IBM WebSphere® Application Server - Express Version 6 and IBM HTTP Server 6.0 in different Windows boxes. So, it is important to specify the fully qualified host name for WebSphere Application Server because it is remote from the HTTP Server point of view. Figure 8-38 shows the text box field of IBM WebSphere® Application Server - Express Version 6 hostname.
- Type the fully qualified hostname of IBM WebSphere® Application Server - Express Version 6 system.
 - Check for the other fields filled in with the default values.
 - Click **Next**.

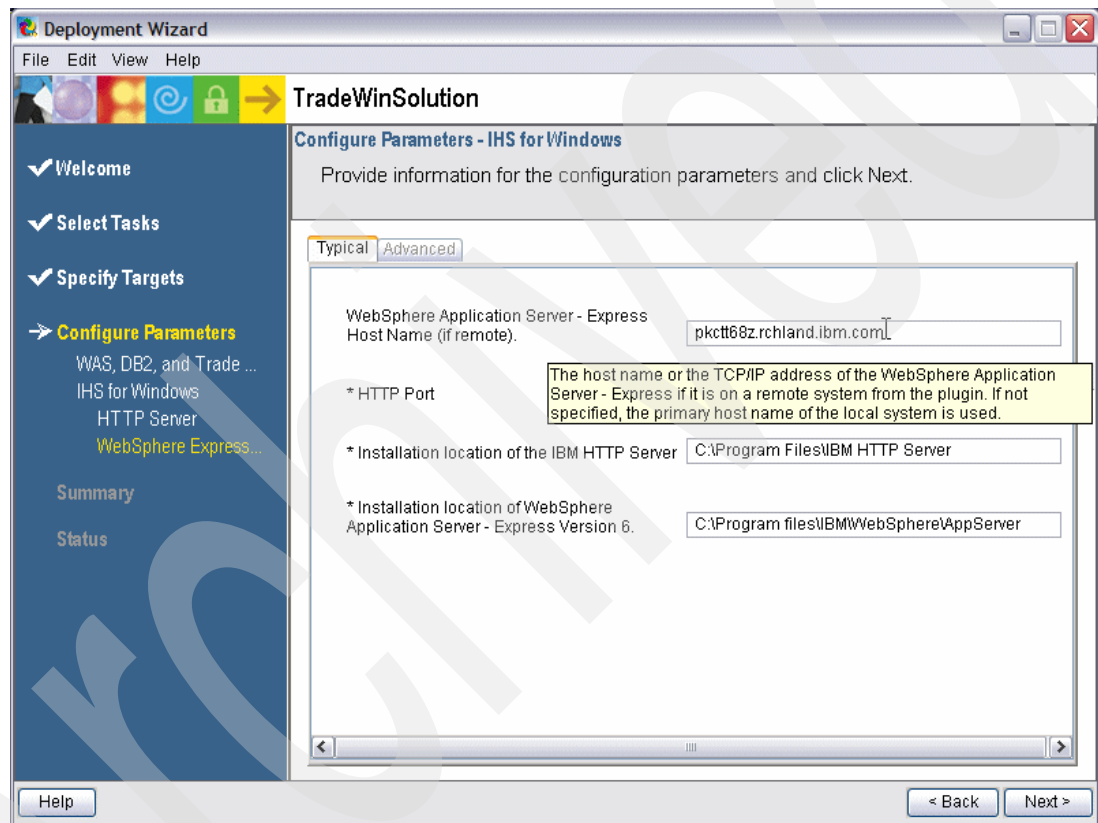


Figure 8-38 Configuring parameter for WebSphere Plug-in

17. The summary panel is shown next. In this example you have two separate tasks, one for each Target Computer. Estimated times for both deployments are displayed on this panel (see Figure 8-39).

Click **Deploy all**. Make sure that IIA has been started on each Target Machine.

Note: Installation time will be less than what is displayed if a middleware component is already installed.

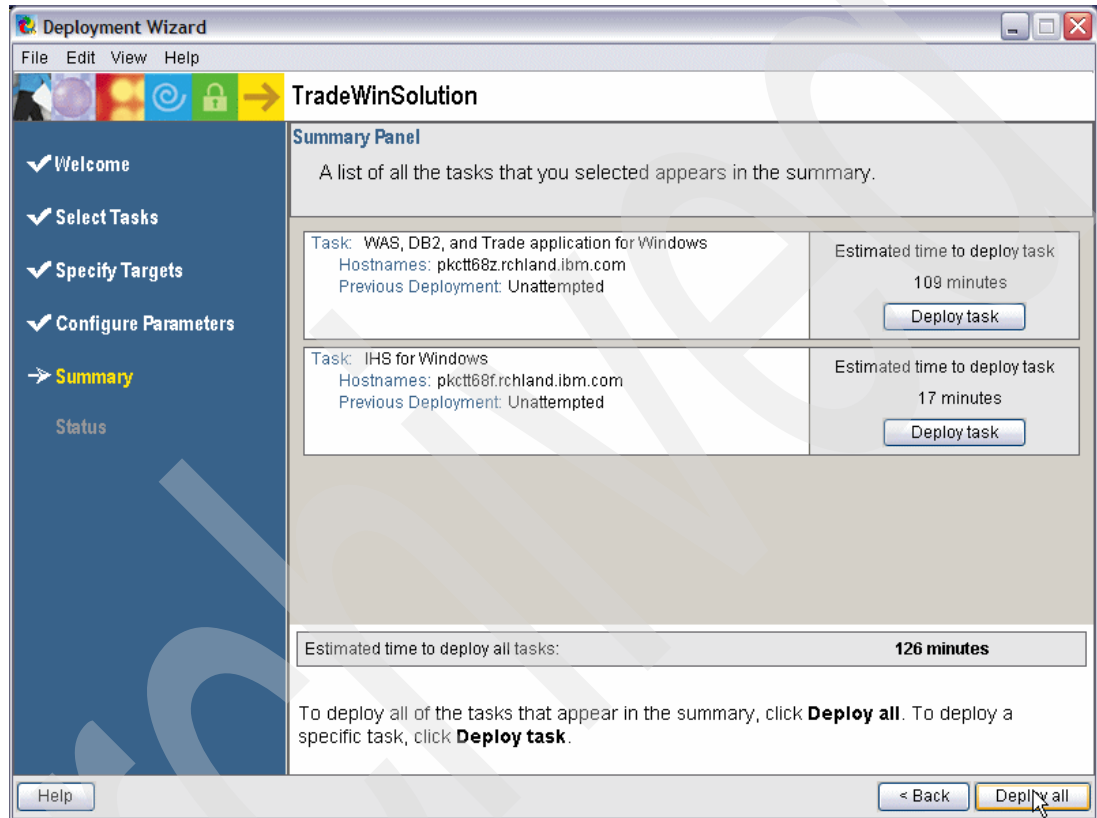


Figure 8-39 Example 2: Summary table of Deployment Wizard

Deployment starts. When all tasks are completed, you should see the messages about the successful deployment components as shown in Figure 8-40.

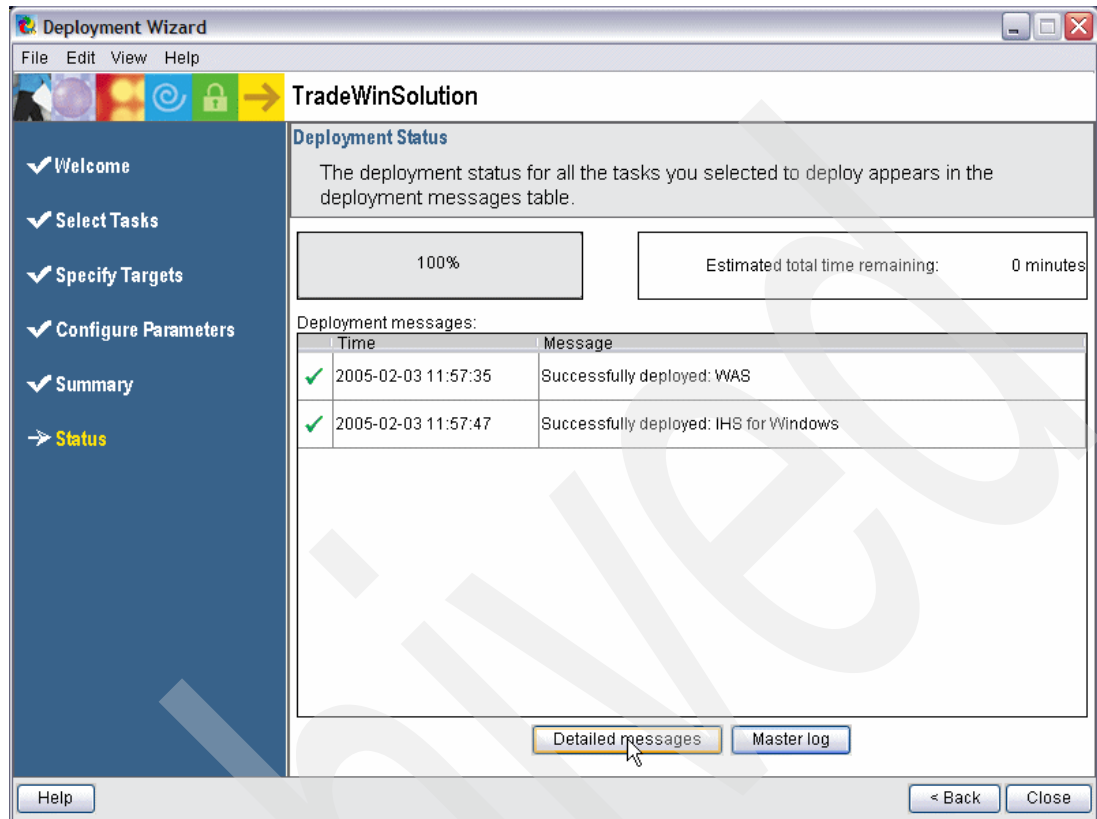


Figure 8-40 Example 2: Successfully Deployment

18. You can also click the **Detailed messages** button. Figure 8-41 shows the messages for each task that is performed. There are also check marks for each task completed successfully and a column that indicates on which system it has been performed.

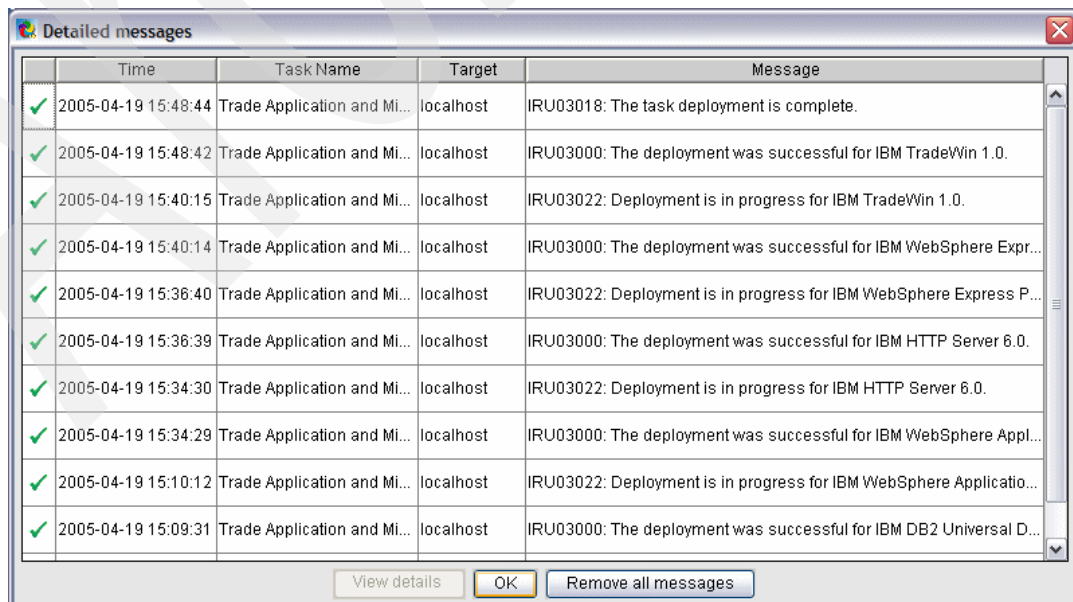


Figure 8-41 Example 2: Deployment Wizard Detailed messages

19.If one or more components indicates deployment errors, open the master log file by clicking the **Master log** button. Look for error messages and resolve the problem. Example 8-4 shows error messages while running the Jacl script.

Example 8-4 Error messages in the Master log

```
**** Starting application install...
**** WASX7017E: Exception received while running file
"/QIBM/userdata/IIA/iru/Flight400_Scripts/WebSphereScript.jacl"; exception information:
com.ibm.websphere.management.application.client.AppDeploymentException: AppDeploymentException: []
**** com.ibm.etools.j2ee.commonarchivecore.exception.OpenFailureException: IWAE0037E Could not open
/tmp/app6456.prop
**** IWAE0037E Could not open /tmp/app6456.prop
**** invalid END header (bad central directory size)
**** com.ibm.etools.j2ee.commonarchivecore.exception.OpenFailureException:
com.ibm.etools.j2ee.commonarchivecore.exception.OpenFailureException: IWAE0037E Could not open
/tmp/app6456.prop
**** WASX7209I: Connected to process "IRAppSvr" on node LPAR3PYM_IRAppSvr using SOAP connector; The type
of process is: UnManagedProcess
**** WASX7303I: The following unrecognized options are passed to the scripting environment and are
available as argv: "[/QIBM/userdata/IIA/iru/Flight400_Scripts/,
/QIBM/userdata/IIA/iru/Flight4004Flight400.prop]"
**** base Node = LPAR3PYM_IRAppSvr
**** Installing application {/QIBM/userdata/IIA/iru/Flight400_Scripts/}...
**** Application Name: /QIBM/userdata/IIA/iru/Flight400_Scripts/
**** Ear file: /QIBM/userdata/IIA/iru/Flight4004Flight400.prop
**** Target Cluster: LPAR3PYM_IRAppSvr
**** Deploy EJB: false
**** Deploy WebServices: false
**** Use default bindings: true
**** Use Ear MetaData: true
**** Starting application install...
**** WASX7017E: Exception received while running file
"/QIBM/userdata/IIA/iru/Flight400_Scripts/WebSphereScript.jacl"; exception information:
com.ibm.websphere.management.application.client.AppDeploymentException: AppDeploymentException: []
**** com.ibm.etools.j2ee.commonarchivecore.exception.OpenFailureException: IWAE0037E Could not open
/tmp/app6456.prop
**** IWAE0037E Could not open /tmp/app6456.prop
**** invalid END header (bad central directory size)
**** com.ibm.etools.j2ee.commonarchivecore.exception.OpenFailureException:
com.ibm.etools.j2ee.commonarchivecore.exception.OpenFailureException: IWAE0037E Could not open
/tmp/app6456.prop
**** Generating plugin
**** Restarting the HTTP server
2005-04-26 09:31:55, 4/mySystem.ibm.com: IRU03018: The task deployment is complete.
2005-04-26 09:31:55, IRU06173: Task 4 in solution C:\Program
Files\IBM\Runtime21\SolutionEnabler\Flight400Solution.ser failed.
```

20. Remote Deployment is now finished. However, in this solution we have IBM HTTP Server 6.0 and IBM WebSphere® Application Server - Express Version 6 on the separate boxes. In order to use the Trade application, you need to copy the plug-in configuration file manually from the IBM WebSphere® Application Server - Express Version 6 system to the IBM HTTP Server 6.0 system. The file is called `plugin-cfg.xml` and is located in the `C:\WebSphere\AppServer\profiles\default\config\cells` directory (this is the default location).

- a. Copy the `plugin-cfg.xml` file from WebSphere system to the `C:\Program Files\IBM HTTP Server\Plugins\config\webserver1` directory on the IBM HTTP Server system.
- b. Edit the `plugin-cfg.xml` file and replace the Log Name tag with the pathname for the `http_plugin.log` file on the HTTP server system. The correct value for the default location of this file is:

```
<Log LogLevel="Error"
Name="C:\PROGRA~1\IBMHTT~1\Plugins\logs\webserver1/http_plugin.log"/>
```

- c. Restart IBM HTTP Server 6.0.

8.5.4 More complex scenarios

We demonstrated three scenarios in this chapter. There are more possible scenarios for deploying a solution. However, the steps you would need to perform will be similar to one of the scenarios.

8.6 What if you have one component installed?

As we mentioned before, a solution is composed of an application and IBM middleware. During the deployment process, we can experience the situation when a middleware component or application is already installed on our target system. There are several variation of the actions that Express Runtime has defined to address this situation.

8.6.1 Distributed platform

For distributed platforms, refer to Windows and Linux systems. Table 8-2 shows possible combinations. The table shows the dependency based on a middleware component and the version of this component already installed on the target system.

Table 8-2 Table of middleware on distributed platform

Middleware	Earlier version	Same version	Later version	Unknown version
DB2 UDB Express We can have only one DB2 UDB per system.	Upgrade is performed automatically by deployment wizard. Important: <ul style="list-style-type: none"> ► For Windows: Perform the DB2 migration steps on any data <i>prior</i> to installing the solution. ► For Linux: This is not so important, since the data can be migrated after the installation completes. 	Deployment wizard prompts with “already exist” messages and skips the installation.	Deployment wizard displays a failure message and terminates the installation. Analyze other applications on the systems that use DB2. Be careful with these actions and analyze the impact for other applications on the system. Run the deployment wizard again to install it.	Deployment wizard displays a failure message and terminates the installation. Uninstall DB2 manually. Be careful with these actions and analyze the impact for other applications on the system. Run the deployment wizard again to install it.
WebSphere Application Server We can run multiple WAS on the same system.	Create a new installation, choosing a different <WAS INSTALL> directory to create a new WAS installation. Modify the port parameters to make sure there are no port conflicts.	<ul style="list-style-type: none"> ► Keep the existing one choosing the existing <WAS INSTALL> directory. ► The Deployment wizard terminates the current installation and uses the existing version. ► Create a new one for our specific reasons like an earlier version. 	Create a new installation like an earlier version.	Create a new installation like an earlier version.
IBM HTTP Server We can run multiple IHS on the same system.	<ul style="list-style-type: none"> ► Create a new installation, choosing a different <IHS INSTALL> directory to continue new IHS installation. ► Modify the port parameters to ensure there are no port conflicts. ► Keep the existing one if version is 2.0 or later. 	<ul style="list-style-type: none"> ► Keep the existing one, choosing the existing <IHS INSTALL> directory. When an earlier version (2.0 or later) of IHS is installed in the target directory, Deployment wizard upgrades it. ► Create a new one for our specific reasons like an earlier version. 	Create a new installation like an earlier version.	Create a new installation like an earlier version.

For example, suppose Deployment Manager tries to install DB2 and an earlier version is already installed. So, we go the second column “Earlier version” and first row “DB2 UDB Express”. The cell at the intersection reads:

Upgrade is performed automatically by deployment wizard.

Warning:

- For Windows: it is important to perform the DB2 migration steps on any data we have PRIOR to installing the solution.
- For Linux: this is not so important, since the data can be migrated after the installation completes

Note: We can have only one DB2 UDB per system. We must pay attention in case of an existing earlier version because the DB2 upgrade is performed by Deployment Wizard. Some DB2 migration steps are required prior (for Windows) or later (for Linux). For more details, refer to the DB2 Web site at:

<http://www-306.ibm.com/software/data/db2/udb/support/>

8.6.2 i5/OS (OS/400) platform

Express Runtime V2.1 comes with pre-built solution wrappers for deployment of IBM WebSphere® Application Server - Express Version 6 and the latest PTFs for IBM HTTP Server 6.0 on OS/400. Wrapper for DB2 UDB Express 8.2 product installation is not required since this product comes natively with i5/OS systems.

Note: We can only have one HTTP product installed on an iSeries. There is a different situation with WebSphere Application Server for iSeries. You can install several *different* versions of the product, but you *cannot* have two installations of the same version.

The following tasks are performed during deployment on i5/OS using Express Runtime V2.1 (there is no DB2 installation task for OS/400):

1. Deployment Wizard applies the latest PTFs IBM HTTP Server (5722DG1) on iSeries.
2. If not installed, Deployment Wizard installs IBM WebSphere® Application Server - Express Version 6 without creating server instances.
3. Deployment Wizard completes the creation of server instances, configuration of the HTTP server instance, and generation of the Web server plug-in.

8.7 Validating the deployment

After the deployment process ends, Deployment Manager displays the status message. You may open the Detailed messages window to check the installation status for each component (see Figure 8-41 on page 222).

In case of an error, you need to look for more detailed information. For this purpose, you should look for the log files.

8.7.1 Log files

We used several abbreviations in this section:

- ▶ <install_path> is the installation pathname that we choose during installation of Express Runtime V2.1
- ▶ <IIA_install_dir> is the pathname used during installation of IBM Installation Agent.

You can check and validate the deployment by viewing the following logs files:

- ▶ **<install_path>/SolutionEnabler/logs/IRU_DeploymentWizard.log.**
This log file contains messages and exceptions that are stored on the Staging Server for both local deployment and remote deployment.
- ▶ **<install_path>/SolutionEnabler/logs/IRU_IITrace.log.**
You can look at this log when you are performing local deployment. This log file shows the print instructions that you have in your user programs (which can aid you when debugging a problem). Data that you write to either the System.out or System.err stream is displayed in this log file. In addition, exceptions that occur while a user program is running also are displayed in this log.
- ▶ **<IIA install dir>/IIA/logs/IRU_IITrace.log.**
This is the same file as above, but located in another directory. You can find it on the Target System when you are performing a remote deployment.
- ▶ **<install_path>/SolutionEnabler/deployment/logs/ibmnsi.log.**
This log file is created when you are performing local deployment. This log file can be used for debugging user programs. It contains the command line call of the most recent user program. You can run this command string from a command prompt to start the user program directly without having to go through the deployment wizard (this may require some additional steps, like setting some variables). In this way, you can invoke your user program from a command line. This may be useful during debugging of your user programs.
- ▶ **<IIA install dir>/IIA/logs/ibmnsi.log.**
This is the same file as above, but on the Target System when you are performing a remote deployment.
- ▶ **<install_path>/SolutionEnabler/deployment/logs/<application_log>.**
This log file is created when you are performing local deployment. The creation of application logs is controlled by the application wrapper and the user programs. Each of the different types of user programs can create a log file, if desired. The logFile attribute in the application wrapper controls the name of the log file.
- ▶ **<IIA install dir>/IIA/deployment/logs/<application_log>.**
This is the same file as above, but located on the Target System when you are performing a remote deployment.

Example 8-5 shows parts of the IRU_DeploymentWizard.log file. Express Runtime V2.1 messages are highlighted. All Express Runtime V2.1 messages start with a 3-character message prefix (IRU) followed by a 5-digit message number. Tokens, such as {0}, {1}, and so on, are used in many messages. These tokens represent computer names, application names, files names, or directory names. The appropriate value is substituted for the token when the message is displayed. Other message codes such as DB21085I or ADMU7701I are specific to the middleware components.

For more details about Express Runtime V2.1 messages, see the Reference and Messages section in the product documentation provided with Express Runtime V2.1.

Example 8-5 Logs extracted from IRU_DeploymentWizard.log.

```
**** IRU10011: Command succeeded.-----
**** Setup Log File Opened... 2005-01-24 11:28:09
**** -----
**** ?== Logging started: 1/24/2005 11:28:18 ==
**** Action start 11:28:18: INSTALL.
**** Action start 11:28:18: AppSearch.
**** Action 11:28:18: AppSearch. Searching for installed applications
**** AppSearch: Property: Y2K_UPDATE2_98, Signature: _Y2K_UPDATE2_98
```

```

**** AppSearch: Property: MDACVERINSTALLED, Signature: _MDACVERINSTALLED
**** AppSearch: Property: IEINSTALLED, Signature: _IESig
**** Action ended 11:28:18: AppSearch. Return value 1.
**** Action start 11:28:18: LaunchConditions.
**** Action ended 11:28:18: LaunchConditions. Return value 1.
**** Action start 11:28:18: FindRelatedProducts.
**** Action ended 11:28:18: FindRelatedProducts. Return value 1..
.
.
.
2005-01-24 11:25:00, 1/localhost: IRU03022: Deployment is in progress for IBM DB2 Universal Database
Express Edition 8.2.
2005-01-24 11:29:59, IRU00013: A communication socket was created on port 1147.
2005-01-24 11:30:11, IRU03015: A data socket was created on port 1150.
2005-01-24 11:32:25, 1/localhost: IRU03000: The deployment was successful for IBM DB2 Universal Database
Express Edition 8.2.
.
.
.
**** 1: Performing input validation:.....Success
**** 1: Configuring DB2 JDBC Applet Service:.....Success
**** 1: Configuring DB2 Security Service:.....Success
**** 1: Setting environment variables:.....Success
**** 1: Initializing instance list:.....Success
**** 1: Creating the DB2 Administration Server:.....Success
**** 1: The value "DB2_EXTSECURITY=YES" was set in the Profile Registry.
**** 1: The value "DB2_ADmingroup=DB2ADMNS" was set in the Profile Registry.
**** 1: The value "DB2_USERSGROUP=DB2USERS" was set in the Profile Registry.
**** 1: Setting default global profile registry variables:.....Success
**** 1: The instance "DB2" has been created successfully.
**** 1: The value "SVCENAME=db2c_DB2" was set in the DBM CFG file for the "DB2"
**** instance.
**** 1: The value "DB2COMM=TCPIP" was set in the Profile Registry for the "DB2"
**** instance.
**** 1: Creating/migrating DB2 instances:.....Success
**** 1: Configuring DB2 Governor Service:.....Success
**** 1: Updating ODBC driver locations:.....Success
**** 1: Registering DB2 licenses:.....Success
**** 1: Installing DB2 performance counters:.....Success
**** 1: Successfully granted complete operating system access to DB2 objects to members
**** of the "DB2ADMNS" group.
**** 1: Successfully granted read and execute operating system access to DB2 objects to
**** members of the "DB2USERS" group.
**** 1: Unable to find the string resource 3575:.....Success
**** 1: Updating global profile registry:.....Success
**** 1: Unable to find the string resource 3574:.....Success
**** 1: Starting all DB2 services:.....Success
**** 1: DB2 Setup wizard has finished copying files to your computer and has completed all the required
system configuration tasks. Please shut down all software programs running on the system now. The programs
can then be restarted and DB2 will be ready for use. The install logs db2wi.log and db2.log are located in
C:\PROGRA~1\IBM\RUNTIM~1\SOLUTI~1\DEPLOY~1\logs\.
.
.
.
**** -----
**** Setup Log File Closed... 2005-01-24 11:31:58
**** -----
2005-01-24 11:32:57, 1/localhost: IRU03022: Deployment is in progress for IBM Console management extension
for DB2 Universal Database 2.1.0.0.

```

```

2005-01-24 11:40:42, 1/localhost: IRU03000: The deployment was successful for IBM Console management
extension for DB2 Universal Database 2.1.0.0.
**** DB21085I Instance "DB2" uses "32" bits and DB2 code release "SQL08020" with
**** level identifier "03010106".
**** Informational tokens are "DB2 v8.1.7.664", "s040914", "WR21342", and FixPak
**** "7".
**** Product is installed at "C:\PROGRA~1\IBM\SQLLIB".
**** IRU10000: The following command was issued:
**** cmd /C C:\iru\IRU_WindowsDB2MgmtExtSetup.exe -is:silent -options
C:\PROGRA~1\IBM\RUNTIM~1\SOLUTI~1\DEPLOY~1\logs\IRU2_1DB2MgmtExt8_2Win4IRU_DB2MgmtExtInstall.iss"
****
**** IRU10011: Command succeeded.(Jan 24, 2005 11:33:49 AM), Install,
com.ibm.jsdt.ismp.wizard.actions.CreateLogEntryAction, msg1, Management Extension, Version 2.1.0.0 for IBM
DB2
.
.
.
**** IRU11009: Starting WebSphere Application Server - Express.
**** IRU10000: The following command was issued:
**** cmd /c C:\WEBSPH~1\APPSE~1\profiles\default\bin\startserver server1
**** IRU11014: Started WebSphere Application Server - Express.
**** ADMU7701I: Because server1 is registered to run as a Windows Service, the
**** request to start this server will be completed by starting the
**** associated Windows Service.
**** ADMU0116I: Tool information is being logged in file
**** C:\WebSphere\AppServer\profiles\default\logs\server1\startServer.log
**** ADMU0128I: Starting tool with the default profile
**** ADMU3100I: Reading configuration for server: server1
**** ADMU3200I: Server launched. Waiting for initialization status.
**** ADMU3000I: Server server1 open for e-business; process id is 2928
**** IRU11009: Starting WebSphere Application Server - Express.
**** IRU10000: The following command was issued:
**** cmd /c C:\WEBSPH~1\APPSE~1\profiles\default\bin\ivt server1 default
**** IRU10011: Command succeeded.
**** Server name is:server1
**** Profile name is:default
**** Profile home is:C:\WebSphere\AppServer\profiles\default
**** Profile type is:default
**** Cell name is:pkctt69tNode01Cell
**** Node name is:pkctt69tNode01
**** Current encoding is:Cp1252
**** Server port number is:9080
**** IVTL0015I: WebSphere Application Server pkctt69t.rchland.ibm.com is running on port 9080 for profile
default
**** IVTL0010I: Connecting to the WebSphere Application Server pkctt69t.rchland.ibm.com on port: 9080
**** IVTL0015I: WebSphere Application Server pkctt69t.rchland.ibm.com is running on port 9080 for profile
default
**** IVTL0015I: WebSphere Application Server pkctt69t.rchland.ibm.com is running on port 9080 for profile
default
**** Testing server using the following
URL:http://pkctt69t.rchland.ibm.com:9080/ivt/ivtserver?parm2=ivtservlet
**** IVTL0050I: Servlet Engine Verification Status - Passed
**** Testing server using the following
URL:http://pkctt69t.rchland.ibm.com:9080/ivt/ivtserver?parm2=ivtAddition.jsp
**** IVTL0055I: JSP Verification Status - Passed
**** Testing server using the following URL:http://pkctt69t.rchland.ibm.com:9080/ivt/ivtserver?parm2=ivtejb
**** IVTL0060I: EJB Verification Status - Passed
**** IVTL0035I: Scanning the file C:\WebSphere\AppServer\profiles\default\logs\server1\SystemOut.log for
errors and warnings

```

```
**** IVTL0040I: 0 errors/warnings were detected in the file
C:\WebSphere\AppServer\profiles\default\logs\server1\SystemOut.log
**** IVTL0070I: IVT Verification Succeeded
...
```

8.8 Troubleshooting deployment

If you encounter errors during the deployment of the solution, you need to identify and resolve an error using the appropriate recovery action. Some of the files that are used during deployment are temporary files. If you want to prevent Express Runtime from deleting temporary files after a deployment, you must use the command line option *-leavefiles*. This option is available for the batch file on Windows platforms (IRU_DebugInstallationAgent.bat) and for the script file on Linux platforms (IRU_DebugInstallationAgent.sh). All options that you can use for debugging are described in Express Runtime V2.1 product documentation.

As an example, this is what you need to do in order to keep the temporary files on the Windows platform:

1. Stop the IBM Installation Agent service if it is running.
2. Start the IBM Installation Agent again using the *IRU_DebugInstallationAgent.bat* file.
3. Start the deployment wizard with the *-leavefiles* option, using the following command:
`<install path>\SolutionEnabler\IRU_TaskInvocation -task deployer -leavefiles`

You can also debug the deployment of your solution. You can setup debugging mode using Deployment Wizard:

1. In the Deployment Wizard (Figure 8-42), select **Edit** → **Preference**.

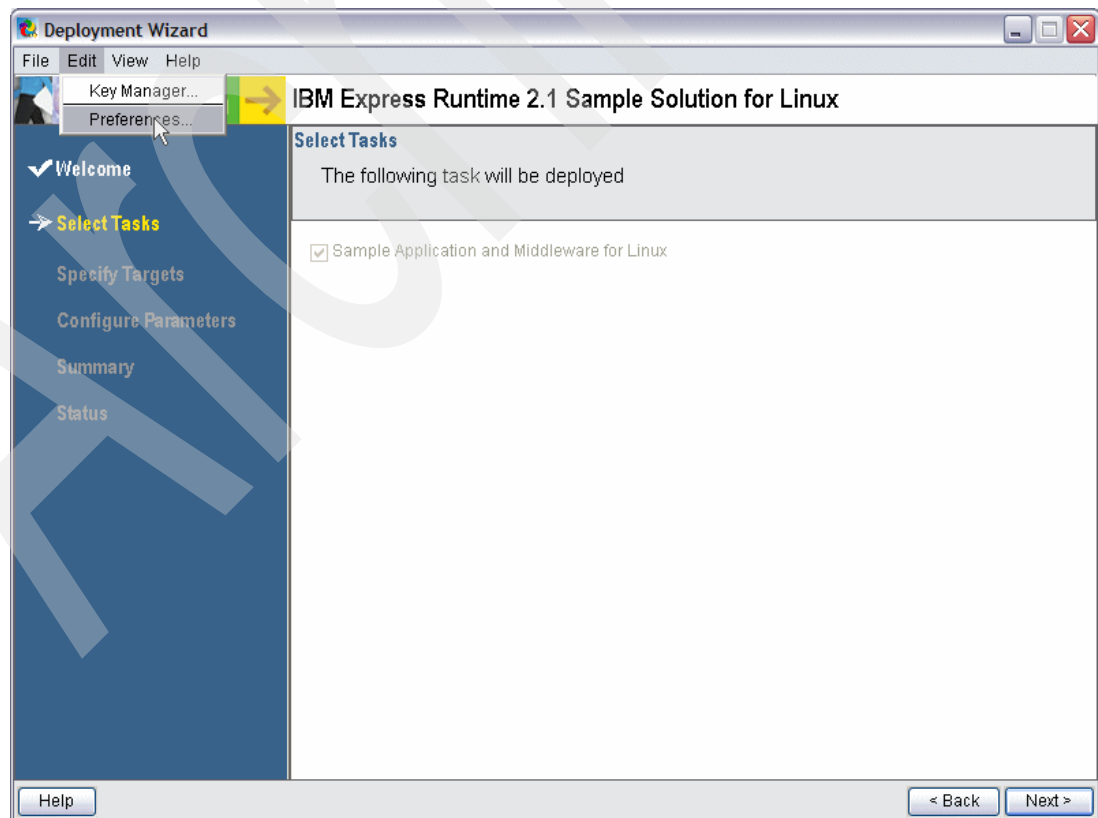


Figure 8-42 Preference of Deployment Wizard

2. Deployment Wizard preferences are displayed. Here you can modify the deployment package path or communication ports. But now, for troubleshooting purposes, click the **Diagnostic Trace** button as shown in Figure 8-43.

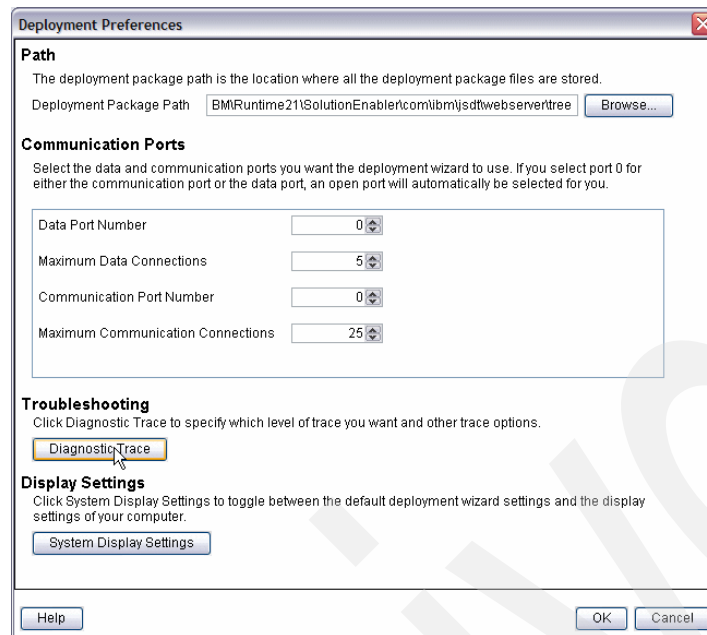


Figure 8-43 Deployment Preferences - choosing Diagnostic Trace

3. The logging capability of Express Runtime V2.1 records support framework diagnostic information and the deployment wizard diagnostic information in separate log files. You can select the desired trace type as shown in Figure 8-44. You can also specify the log file name and the maximum amount of space for each log file.

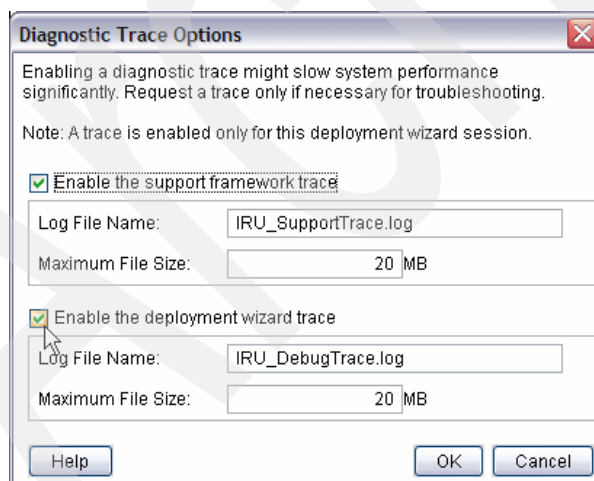


Figure 8-44 Enable Diagnostic Trace Options

There are some differences in what type of information is logged in each file:

Deployment Wizard trace provides a trace of internal deployer code that is executed during a deployment. This trace is of little value to a customer doing a deployment.

Support framework trace provides a method level trace of all support framework methods called on the target machine during deployment (the target machine is the same machine as the staging server during a local install). This trace also includes any trace statements that the user has included in their user programs. This is the trace mechanism that customers may be interested in.

Note: Logging diagnostic information is CPU intensive, and therefore slows the system. Use diagnostic logging only when it is necessary for troubleshooting of the deployment process.

8.8.1 Tracing on the target system

To enable diagnostic trace on a remote computer, start the agent with trace options:

```
<IIA install_directory>\IIAJRE\bin\java -jar DJT_ibmnsit.jar -task intallationAgent  
-enableSupportFrameworkTrace -enableSolutionDeployerTrace
```

To run the trace-enabled agent in debugging mode, add the *-leavefiles* option to the end of the above command.

On OS/400 you can run this task from the QShell command line as follows (enter on one line):

```
/QIBM/ProdData/IIA/IRU_ia_start-agent -enableSupportFrameworkTrace  
-enableSolutionDeployerTrace -leaveFiles
```

Managing Express Runtime

This chapter discusses the topic of managing the middleware components from a browser-based console.

The common way to administer middleware components deployed with Express Runtime V2.1 is by using Console for Express Runtime, which allows you to perform many administrative tasks for every middleware component using the same user interface. If you are a system administrator or software developer, Console for Express Runtime can help you by providing a Web-based central location to perform many middleware management and administration tasks

In this chapter we introduce Console for Express Runtime V2.1 and describe how to install and use it.

The following topics are discussed:

- ▶ Integrated Solutions Console
- ▶ Console for Express Runtime
- ▶ Deploying Console for Express Runtime
- ▶ How to use Console for Express Runtime
- ▶ Other ways to administer Express Runtime V2.1 middleware components

9.1 Console for Express Runtime

Using the Console for Express Runtime is the common way to administer IBM middleware provided and deployed with Express Runtime V2.1. The Console for Express Runtime provides a single, Web-based interface for performing administrative tasks. Thus, the Console for Express Runtime simplifies the experience of managing the IBM middleware, such as DB2 UDB Express 8.2, IBM WebSphere Application Server - Express Version 6, WebSphere Plug-in, and IBM HTTP Server 6.0.

The Console for Express Runtime helps you to manage multiple instances of each middleware component. These middleware components can be on one or more computers. We can use Console for Express Runtime to perform the following administrative tasks:

- ▶ Check a server status
- ▶ Start and stop application servers, Web servers, and DB2 databases
- ▶ Perform a one-step database backup
- ▶ Configure log settings and view logs
- ▶ Modify IBM WebSphere Application Server - Express Version 6 configuration settings

More details about administrative tasks can be found in the product documentation installed with Express Runtime V2.1.

9.1.1 Terminology

When you access console for Express Runtime, you may notice two distinct product names: *Integrated Solutions Console V5.1 (ISC)* and *Console for Express Runtime V2.1*. The Integrated Solutions Console is like a framework where you install the console for Express Runtime specific features. Integrated Solutions Console is based on WebSphere Portal Server V5.0.2.2 that is installed automatically during deployment of the console. Integrated Solutions Console provides the Console layout, and the service of logging and navigating the console via a Web browser. Console for Express Runtime is the set of Portal applications installed into ISC. In this chapter we refer to Console for Express Runtime every time we talk about the console for managing Express Runtime V2.1 middleware components.

9.2 Deploying Console for Express Runtime

Before you deploy Console for Express Runtime, ensure that the hardware and software requirements are met for the target computers to which you deploy. The Integrated Solutions Console can be deployed from any OS supported by Express Runtime V2.1. Refer to Table 8-1 on page 178 for details.

Note: We recommend deploying Console for Express Runtime on a dedicated system. The deployment task of Console for Express Runtime creates a WebSphere Portal Server V5.0.2.2 instance with application portlets for managing the middleware. So it is better to deploy Console for Express Runtime on a separate box for performance reasons.

9.2.1 Example of the remote deployment

You can deploy Console for Express Runtime on a Windows or Linux system. As we described in 8.1, “Supported platforms” on page 178, OS/400 box cannot be a Staging Server. Also, the console can’t be deployed to an OS/400 or i5/OS system. However, these systems can be managed using the console.

In this example we show the step-by-step process of deploying Console for Express Runtime for Linux from a Windows Staging Server:

1. On the Target Computer, you have to start IBM Installation Agent now. Log to the Linux system as root and perform the following commands:

```
[root@relinux root]# cd /opt/IBM/IIA/  
[root@relinux IIA]# ./IRU_ia_start-agent
```

More details about using IBM Installation Agent can be found in 8.4, “Installing the IBM Installation Agent (IIA)” on page 183.

Note: The IBM Installation Agent on the Target Computer must be started before performing remote deployment. In addition, you must set up the same secure phrase on both systems. More details about the Secure key are described in 8.5, “Deployment scenarios” on page 195.

2. On the Staging Server, click **Start** → **Programs** → **IBM Express Runtime 2.1** → **Deployment Wizard**. A window similar to Figure 8-15 on page 197 is displayed.
3. For starting deployment process open the serialized (.ser) file. Click **File** → **Open** and choose **IRU2_1MiddlewareAll.ser** from default directory.
4. Click **Next**.
5. Click the checkbox for console for Express Runtime as shown in Figure 9-18 and click **Next**.

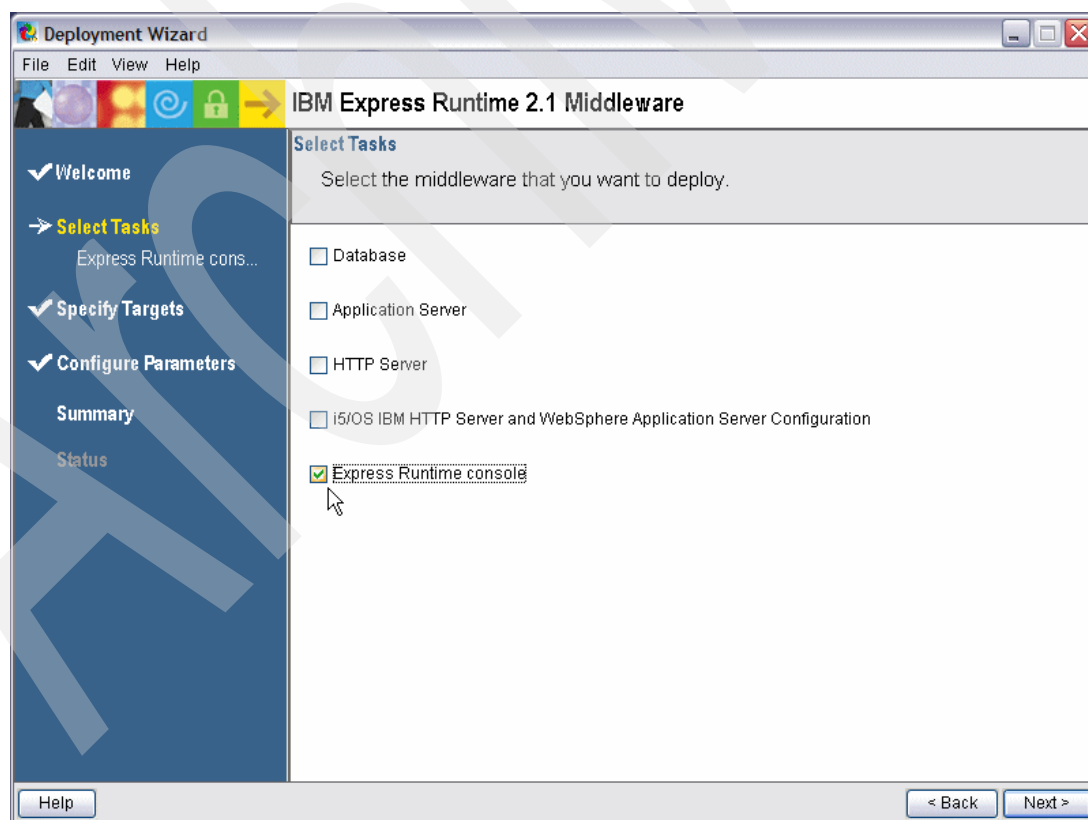


Figure 9-1 Deploying console for Express Runtime

6. In the next panel, select the version of console for Express Runtime as shown in Figure 9-2. Select the checkbox for **Express Runtime Console 2.1 for Linux**. Click **Next**.

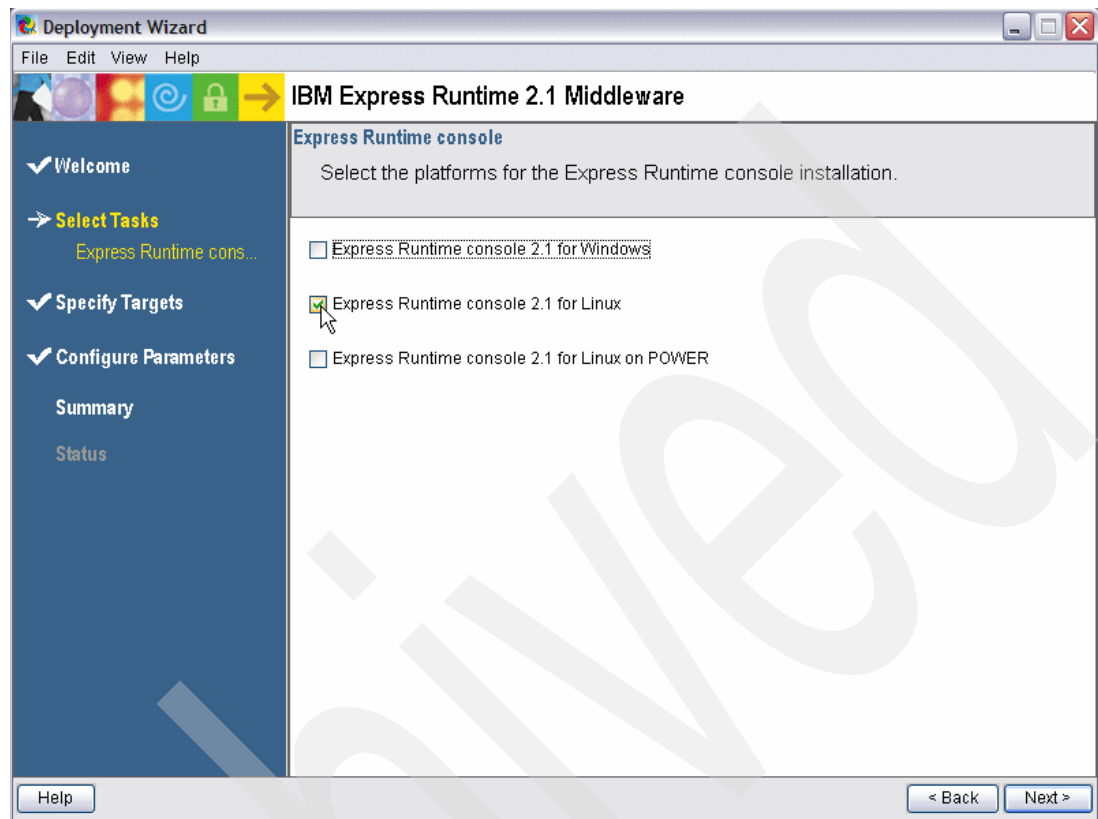


Figure 9-2 Deploying console for Express Runtime for Linux

7. In the next panel you have to specify the target computer where you want deploy Console for Express Runtime.
- In our example, we type `relinux.rchland.ibm.com` in the target computer field and click **Add**. You should use your own value.
 - Click the **Test connections** button for testing the connection.

Note: In our example, `relinux.rchland.ibm.com` is the fully qualified hostname of Linux Target Computer.

8. A pop-up window (see Figure 8-33 on page 215) is displayed showing the test results.

9. In the next panel, you must provide the configuration parameters for the console for Express Runtime. The target directory and some other fields are filled in with the default values, but you can change them. Provide the fully qualified host name as shown in Figure 9-3. In our example, we type `relinux.rchland.ibm.com` in the hostname field. Click **Next**.

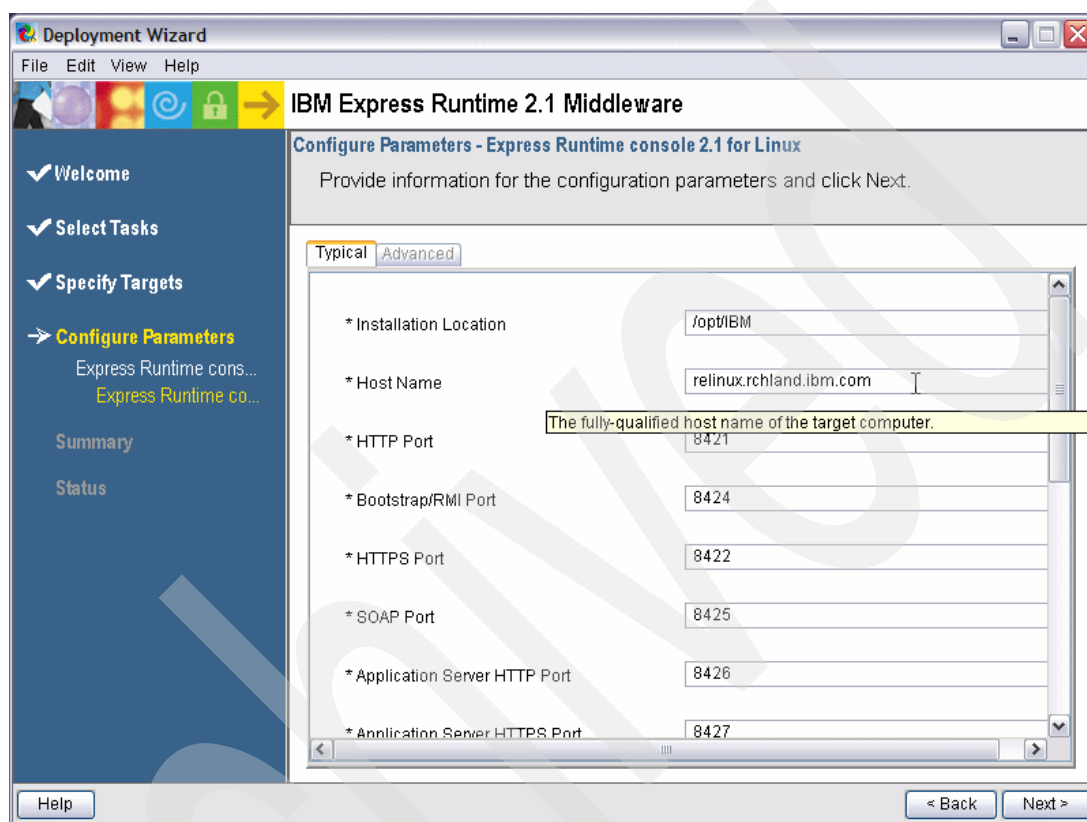


Figure 9-3 Configuration parameter for console for Express Runtime

10. The Summary table is shown next. In this example you have only one task, so you can click either **Deploy all** or **Deploy Task**. An estimated time for the deployment process is displayed.

Note: Remote deployment could take a lot of time. In our example it took about five hours.

11. Deployment starts. A status bar and estimated total time remaining are displayed.

12. When installation completes, a message of successful deployment is displayed as shown in Figure 9-4.

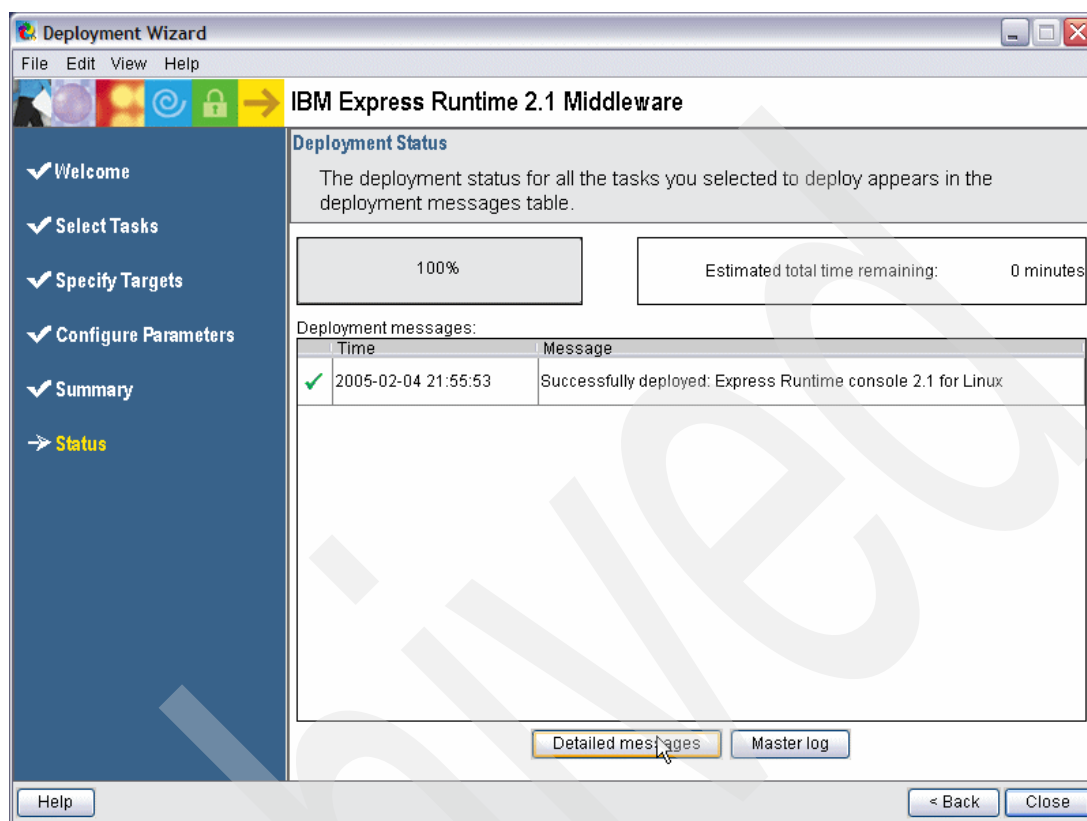


Figure 9-4 Successfully deploy of Console for Express Runtime

13. You can click **Master log** to open the IRU_DeploymentWizard.log file located in C:\Program Files\IBM\Runtime21\SolutionEnabler\logs (for this example) as shown in Figure 9-5.

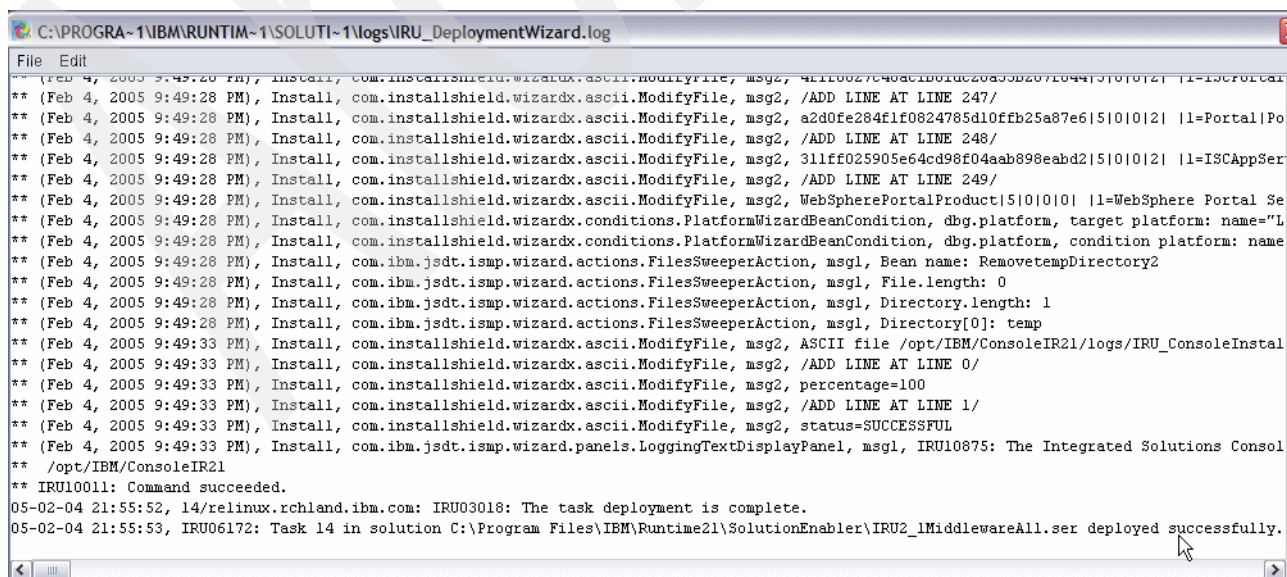
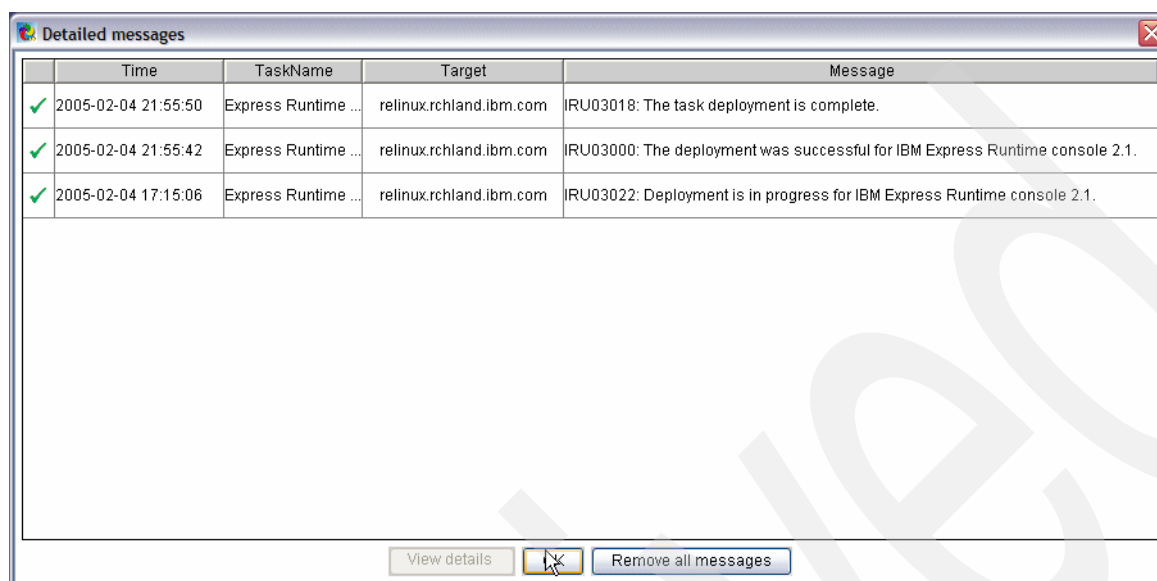


Figure 9-5 Viewing Master log for Deploying Console for Express Runtime

14. Close the log file window and click **Close** in the Deployment Wizard window.
15. You can also click **Detailed Messages** to view the status messages for each component.
Figure 9-6 shows the three steps for remote deployment of Console for Express Runtime.



	Time	TaskName	Target	Message
✓	2005-02-04 21:55:50	Express Runtime ...	relinux.rchland.ibm.com	IRU03018: The task deployment is complete.
✓	2005-02-04 21:55:42	Express Runtime ...	relinux.rchland.ibm.com	IRU03000: The deployment was successful for IBM Express Runtime console 2.1.
✓	2005-02-04 17:15:06	Express Runtime ...	relinux.rchland.ibm.com	IRU03022: Deployment is in progress for IBM Express Runtime console 2.1.

View details [icon] Remove all messages

Figure 9-6 Detailed messages for deploy Console for Express Runtime

9.3 Using Console for Express Runtime

To use Console for Express Runtime, you need to:

- ▶ Start IBM Installation Agent (IIA) on each target system. IIA is configured as a service (by default) and it starts when you boot your system.
- ▶ Make sure the management extension components are installed along with the corresponding middleware. These additional components (management extensions) provide the support for Console for Express Runtime. Look at Figure 5-19 on page 83. It shows several components with the *MgmtExt* suffix. These are the management extension components.

Note: You can administer only middleware installed via Express Runtime V2.1.

9.3.1 Logon to Console for Express Runtime

To access Console for Express Runtime, follow these steps:

1. Open a Web browser.
2. For our example, we point the browser to the following URL:
`http://relinux.rchland.ibm.com:8421/ibm/console`
3. The server switches to the secure URL for login in:
`https://relinux.rchland.ibm.com:8422/ibm/defaultconsole/!ut/p/.scr/Login`
Note that it changes the TCP protocol and port for secure connections.
4. Specify **username** and **password** (see Figure 9-7). The first time, you have to log in with the user ID and password that you specified during the deployment time.

5. Click **Log in**.

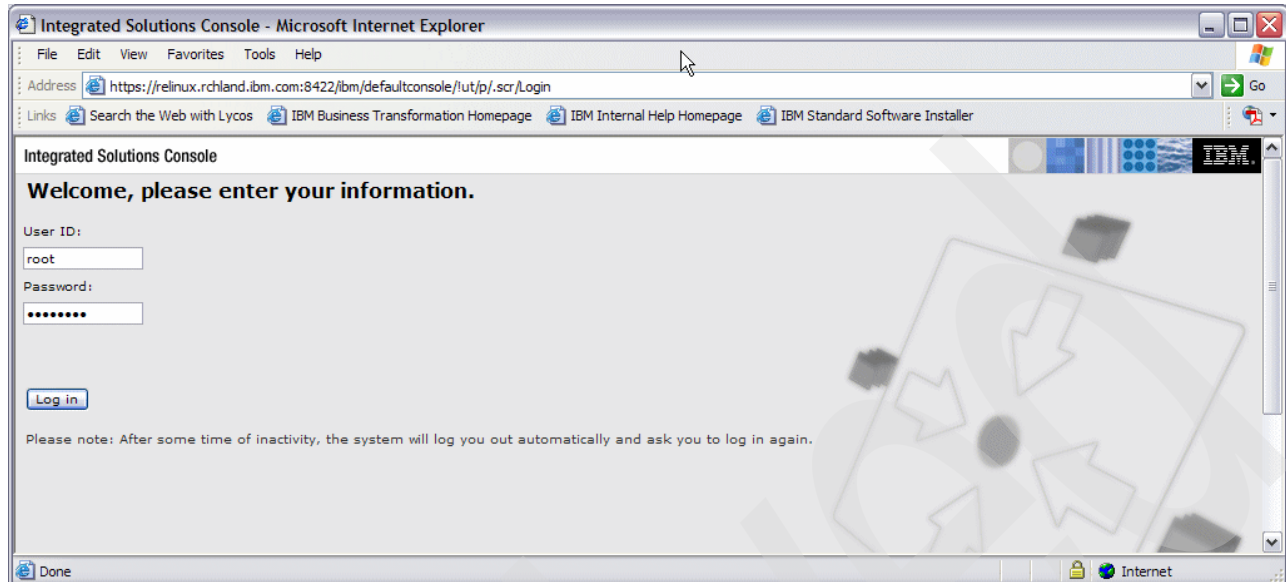


Figure 9-7 Login on Console for Express Runtime

6. Now you are logged in to Console for Express Runtime as shown in Figure 9-8.

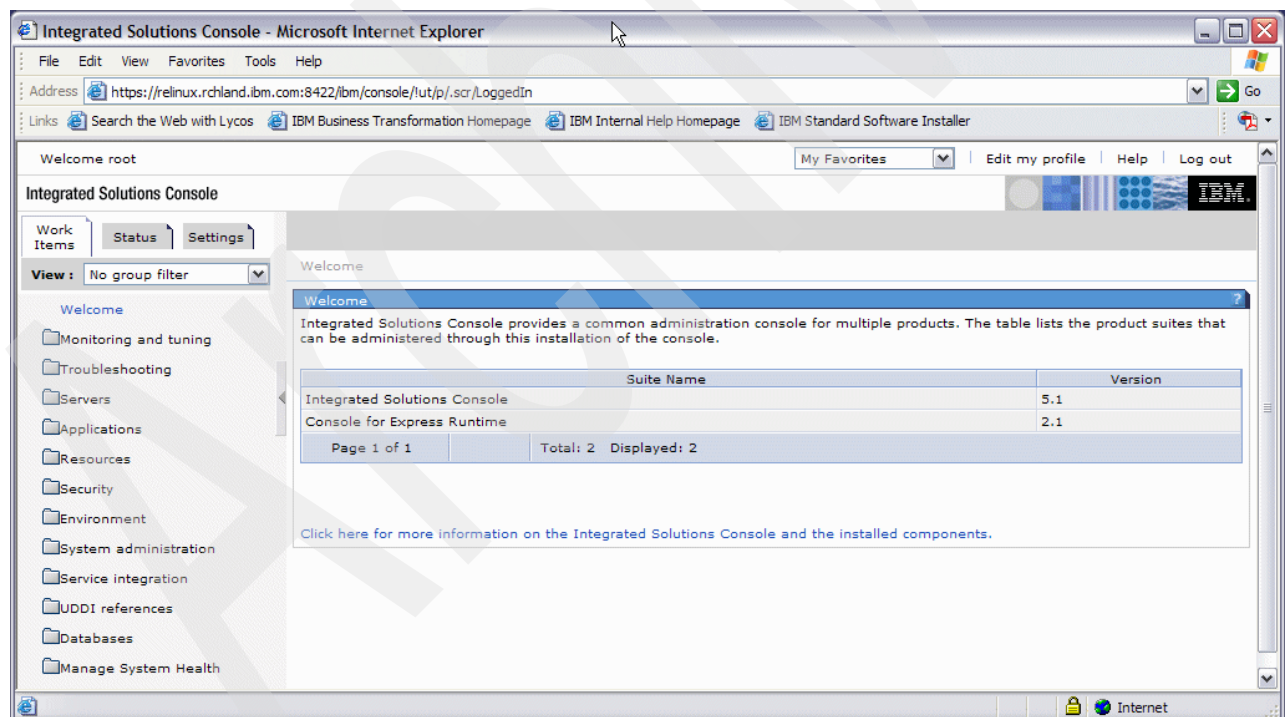


Figure 9-8 Console for Express Runtime - Web home page

The layout of Console for Express Runtime is provided by Integrated Solutions Console. You have three options in the navigation tabs: Work Items, Status, and Setting. For each one, you have a navigation tree on the left with the list of all the administration tasks that you can perform.

9.3.2 Adding a server to the console

Before you can administer a server, you need to add it to the list of managed servers. You also need to perform a test connection to this server. It does not matter if the server that you want administer is an IBM HTTP Server 6.0 or an IBM WebSphere Application Server - Express Version 6 or DB2 UDB Express 8.2. You have to do the same operation for every server that you want to administer. Now we show you how to add an IBM WebSphere Application Server - Express Version 6 in the list of administered servers:

1. Click **Servers** in the navigator tree as shown in Figure 9-8.
2. Click **Application Server**.
3. Click **Status - application server**.
4. Now click **Add / Remove servers** in the central work area as shown in Figure 9-9.

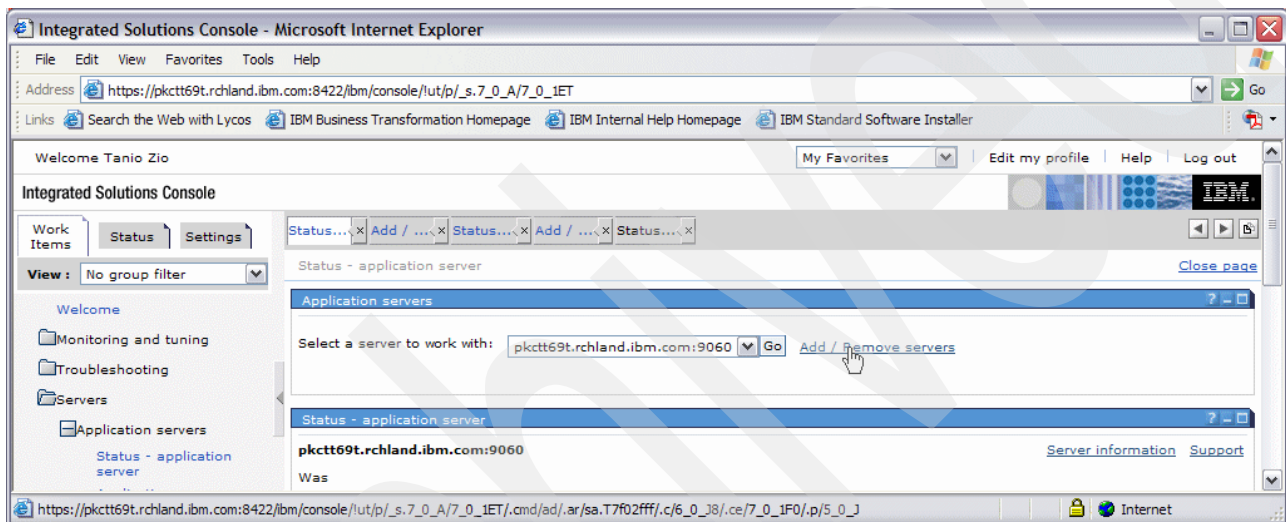


Figure 9-9 Console for Express Runtime - Add / Remove server to administer -

5. Enter the following information (Figure 9-10):
 - a. The fully qualified host name
 - b. The http port of your WebSphere Application Server
 - c. The console agent port
6. Click **Add to list**.

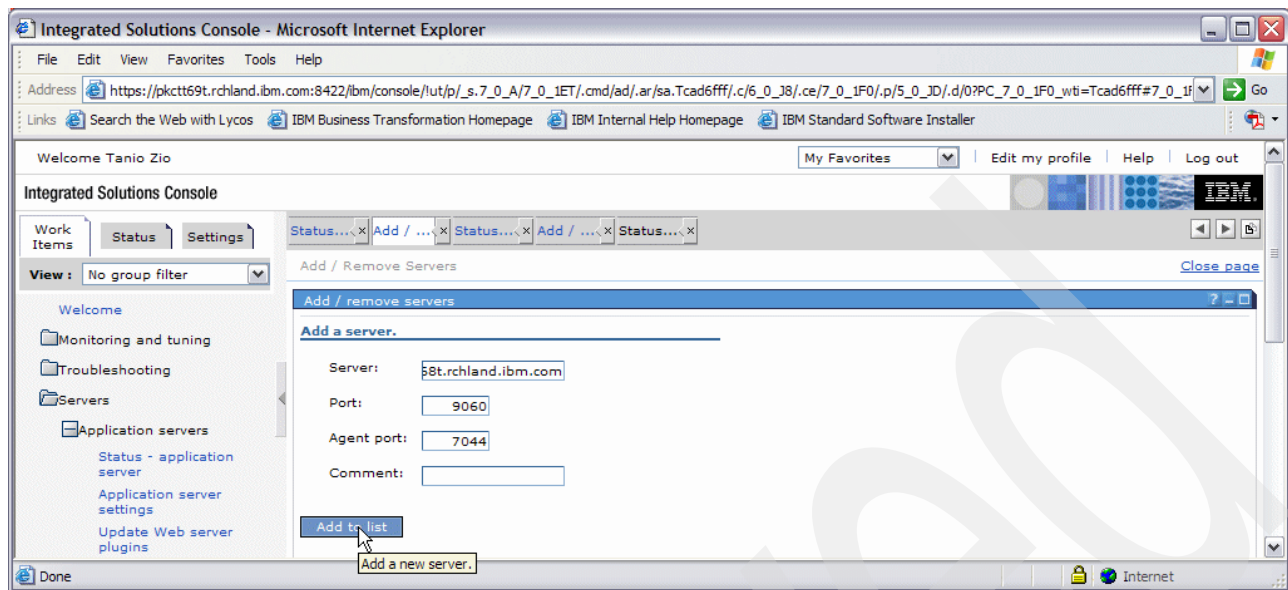


Figure 9-10 Console for Express Runtime - Add server to administer

7. The server is now added to the list of managed servers. You can select it and click **Test connection**.
8. Now provide a valid username and password for IBM WebSphere Application Server - Express Version 6 box and click **Test** as shown in Figure 9-11.

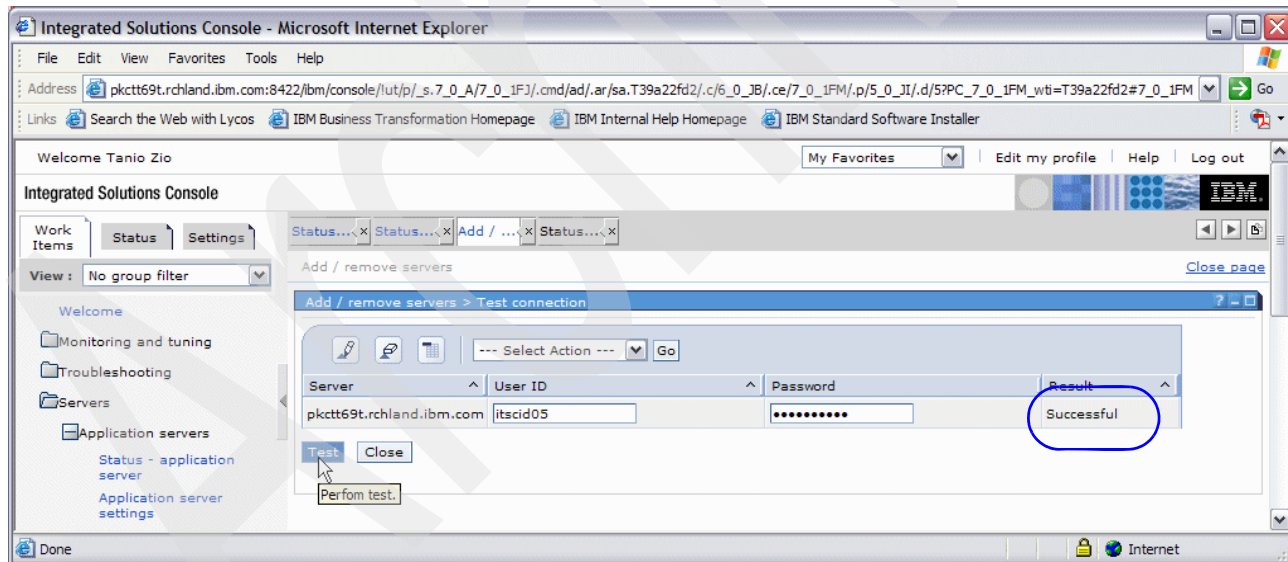


Figure 9-11 Console for Express Runtime - Test Connection

9. The results of the test are displayed in the result column (see Figure 9-11).

You should perform similar steps for IBM HTTP server and DB2 server. The only difference is that you should select a corresponding server in the navigation tree.

9.3.3 Starting or stopping a server

After you have added a server to the list of administered servers, you can start or stop this server using Console for Express Runtime. It does not matter if the server that you want administer is an IBM HTTP Server 6.0 or an IBM WebSphere Application Server - Express Version 6 or DB2 UDB Express 8.2. You have to do the same operation for every server that you want to administer. Now we show you how to use Console for Express Runtime for stopping a WAS server:

1. Click **Monitoring and tuning** (see Figure 9-8 on page 240).
2. Click **all application servers**.
3. In the working area of your browser, the list of application servers is displayed showing the actual operational status.
4. Click the checkbox for a desired server.
5. Click **Stop** as shown in Figure 9-12.

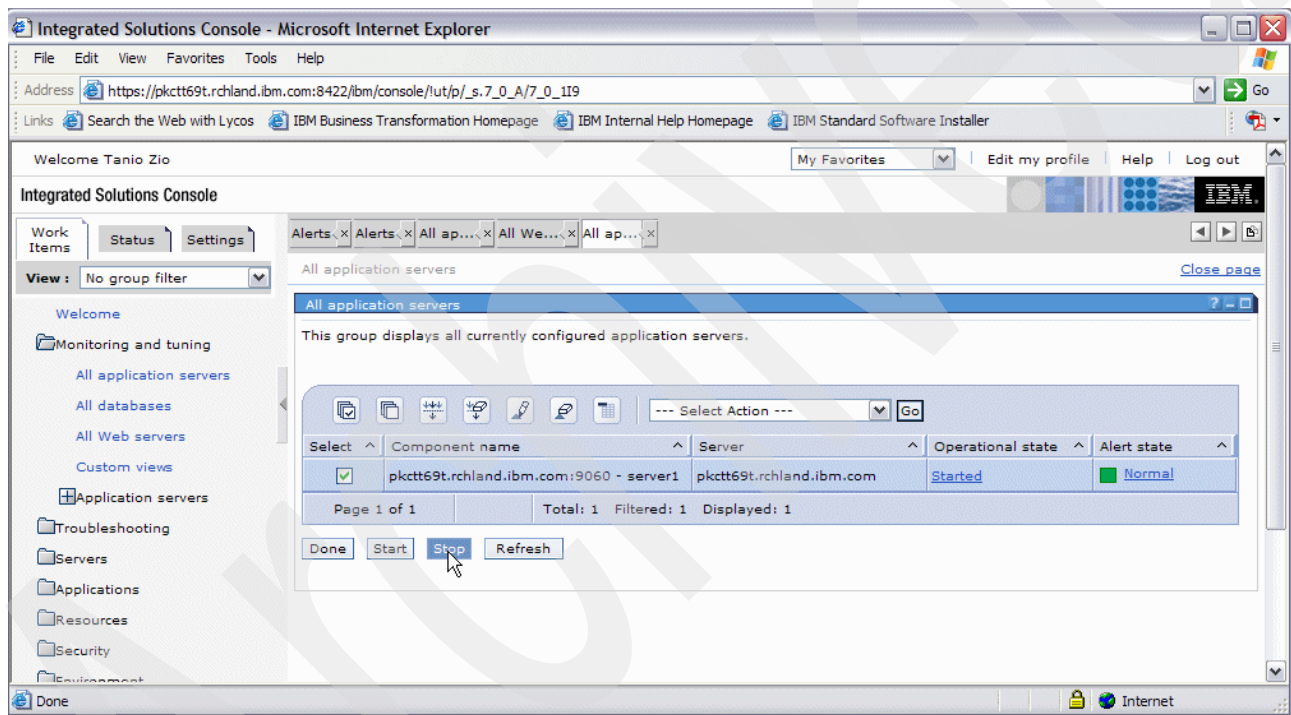


Figure 9-12 Console for Express Runtime - Stopping IBM WebSphere Application Server - Express Version 6

6. You should confirm your action: Click the **OK** button to stop IBM WebSphere Application Server - Express Version 6.
7. Only when the selected server is stopped, then Console for Express Runtime returns. It displays the actual operational state (see Figure 9-12).

9.3.4 Viewing log files

You can view log files of IBM WebSphere Application Server - Express Version 6 and IBM HTTP Server 6.0 using Console for Express Runtime. It does not matter if the server that you want administer is an IBM HTTP Server 6.0 or an IBM WebSphere Application Server - Express Version 6. You have to do the same operation for both kinds of servers that you want to administer. Now we show you how to view IBM HTTP Server 6.0 log files using Console for Express Runtime:

1. Click **Troubleshooting** (see Figure 9-8 on page 240).
2. Expand **Web Servers**.
3. Click **Log viewer**.
4. Select the HTTP server instance in the central work area of your browser and click **OK**.
5. Select which file you want to view (in the drop-down menu) and click **GO** as shown in Figure 9-13.

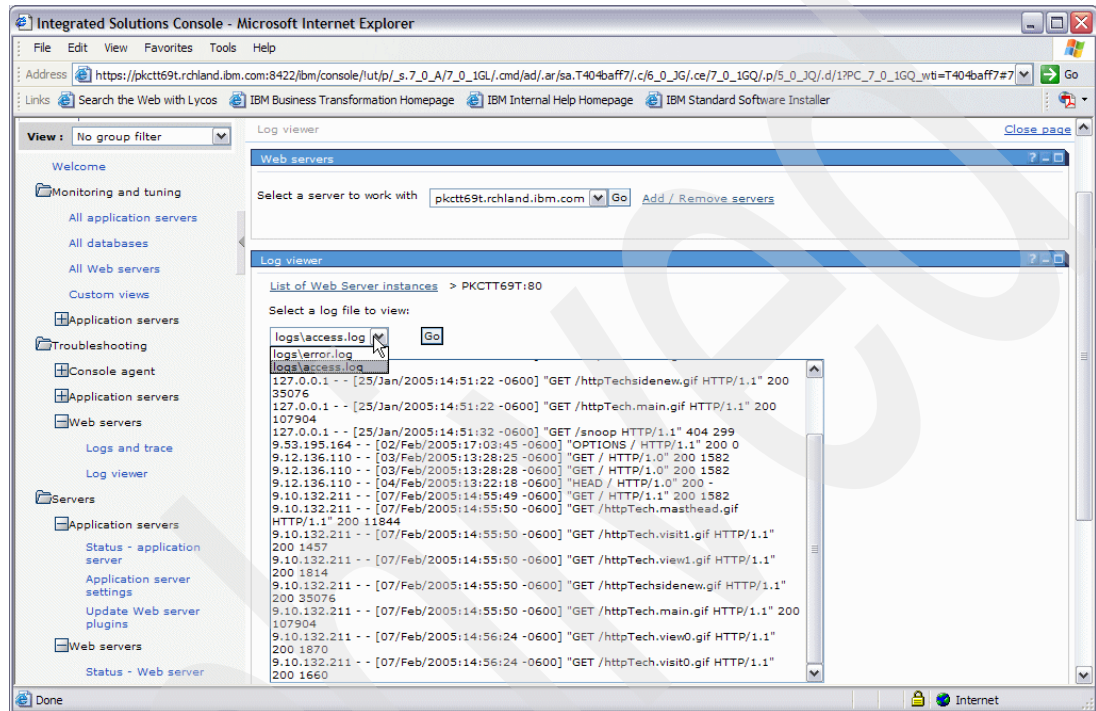


Figure 9-13 View log files with Console for Express Runtime

6. The log file selected is now displayed in your browser.

9.3.5 Backing up databases

You can also perform some administrative tasks for DB2 UDB Express 8.2 databases. In the navigation tree, expand **Databases** and **Manage System Health**. Figure 9-14 shows the options that you can perform for database administration. You can click **Databases** and then **Alerts** to see if there is any alert.

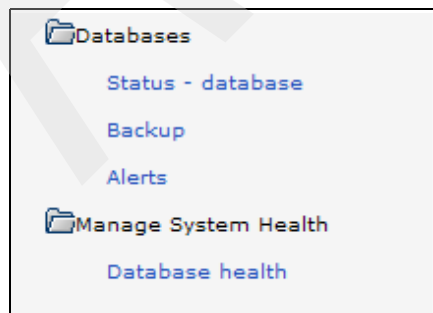


Figure 9-14 Console for Express Runtime - Databases options on navigation tree menu

As an example, we perform a one-step backup of a selected database on our Windows box:

1. Click **Databases** in the navigation tree (see Figure 9-8 on page 240).
2. Click **Status - database**.
3. Click **Add / remove databases** in the central work area of your browser.
4. Enter valid values for the following fields as shown in Figure 9-15:
 - **shura.rchland.ibm.com** (DB2 server)
 - **8888** (JMX port)
 - **7044** (Agent port)
 - **DB2** (Instance)
 - **TRADEDB** (Name of Database)

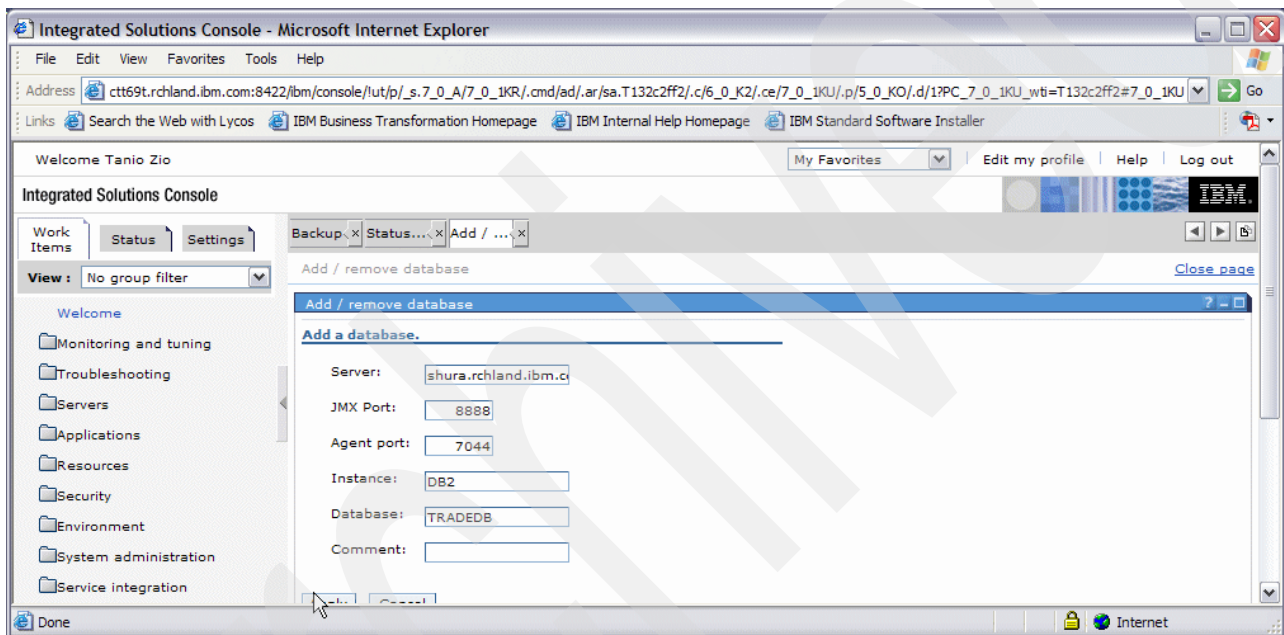


Figure 9-15 Console for Express Runtime - Add database

5. Click **Add to list**.
6. In the navigate tree, click **Backup**.
7. In central work area, select a directory to store a backup copy. For example, type C:\backupdb2 and click **OK**.

Note: Backup will be performed on the DB2 server. The directory specified must exist on the DB2 server box.

8. The backup starts. A message as shown in Figure 9-16 is displayed in your browser.

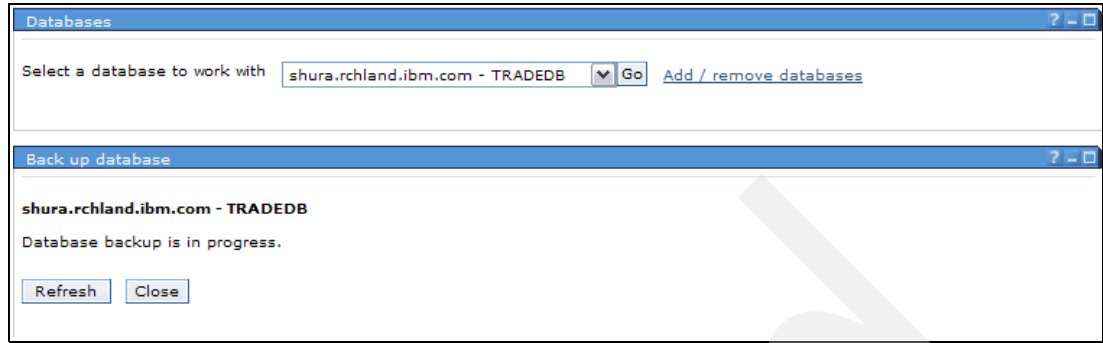


Figure 9-16 Console for Express Runtime - DB2 backup in progress

9. After a while, click **Refresh** to view the status of a backup. When the backup completes, a message about successful backup is displayed (see Figure 9-17).

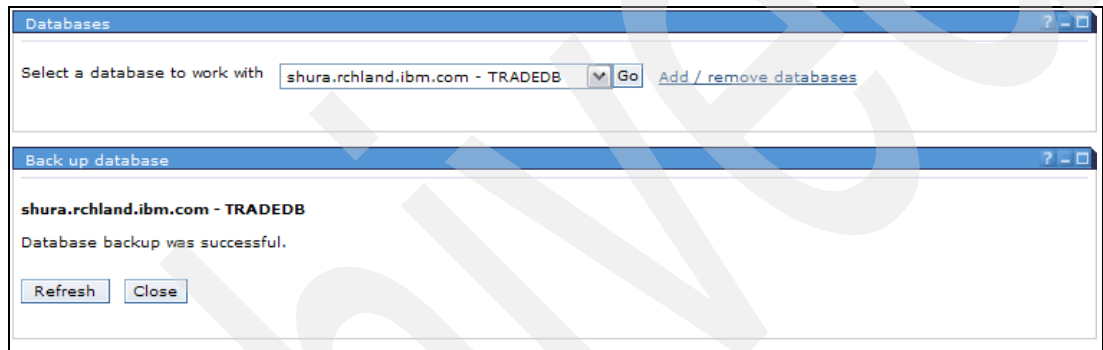


Figure 9-17 Console for Express Runtime / backup db2 completed

9.4 Other ways to manage Express Runtime middleware

In this section we point to other tools to manage Express Runtime V2.1 middleware components. Each tool is specific to a middleware component.

9.4.1 IBM HTTP Server 6.0 on Windows and Linux

To administer and configure IBM HTTP Server 6.0 on Windows and Linux systems, you can use the IBM HTTP administration server that is installed within the IBM HTTP Server 6.0 Administration function for IBM HTTP Server 6.0. Refer to the InfoCenter Web site for more details:

http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.ihs.c/info/welcome_ihs.html

9.4.2 IBM HTTP Server on OS/400

In order to administer and configure the IBM HTTP Server on OS/400, you have to start the *ADMIN instance of IBM HTTP Server. By default, you access this interface through your Web browser on port 2001. For example:

`http://rchasral.rchland.ibm.com:2001`

For more information about administration of IBM HTTP Server 6.0, you may refer to Chapter 6 of the redbook, *WebSphere Application Server V5 for iSeries*, SG24-6588.

9.4.3 IBM WebSphere Application Server - Express Version 6

For IBM WebSphere Application Server - Express Version 6, you can use the WebSphere Administrative Console. This tool is installed during deployment task of IBM WebSphere Application Server - Express Version 6 by default. It is a J2EE application that allows you to perform any administrative task on WebSphere Application Server. You can use the administrative console to create and manage resources, applications, and servers or to view product messages.

Refer to the InfoCenter Web site at:

<http://publib.boulder.ibm.com/infocenter/wasinfo/index.jsp?topic=/com.ibm.websphere.exp.doc/info/exp/ae/welcadminconsole.html>

9.4.4 DB2 UDB Express 8.2 for Windows and Linux

For administration tasks of DB2, you can use a specific product tool called *DB2 Control Center*. It is installed by default during the deployment task of DB2 on your Target Computer. It is a graphical user interface (GUI) available on both Windows and Linux platforms. You can start DB2 Control Center from Windows and Linux in the following ways:

- ▶ On Windows: Click **Start** → **Programs** → **IBM DB2** → **General Administration Tool** → **Control Center**
- ▶ On Linux, perform the following steps:
 - a. Connect to Linux GUI console as DB2 instance name (for example, db2inst).
 - b. Click the **IBM DB2** folder.
 - c. Click the **Control Center** icon.

In both environments, the same graphical window is displayed as shown in Figure 9-18.

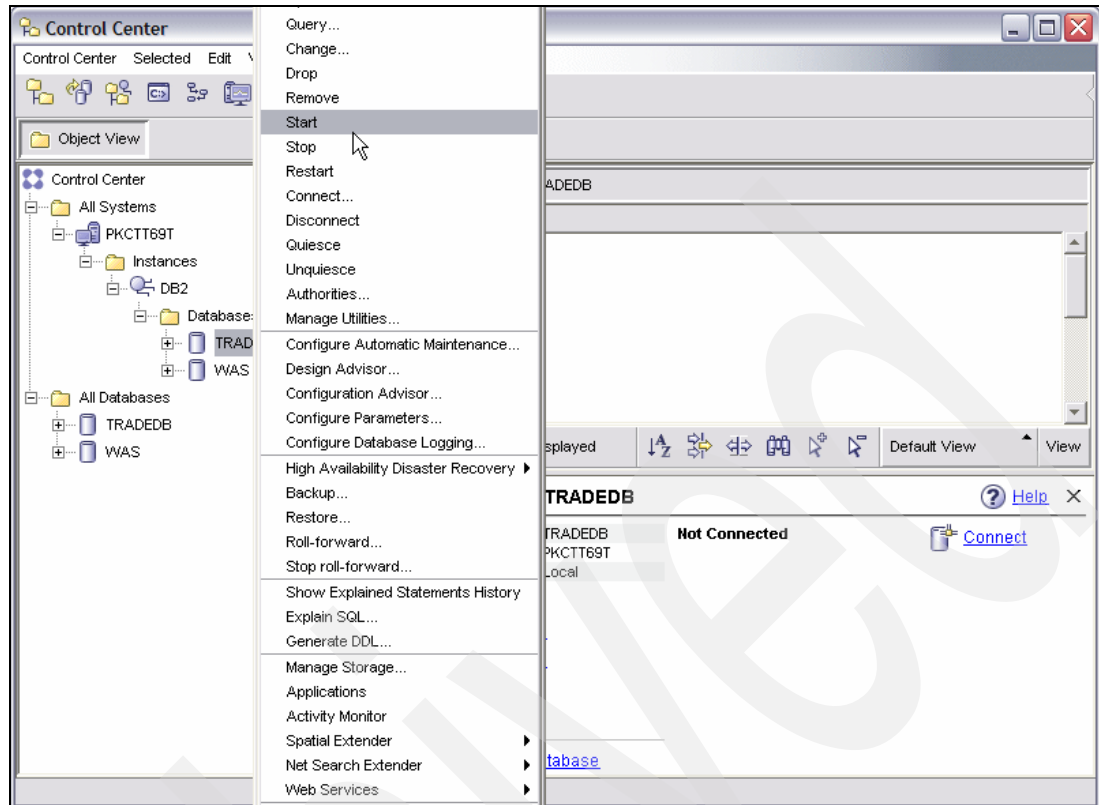


Figure 9-18 DB2 Control Center - Managing a database

With DB2 Control Center, you can start and stop a database and perform all tasks that are shown in Figure 9-18. For details about DB2 Control Center, refer to the DB2 InfoCenter at:

<http://publib.boulder.ibm.com/infocenter/db2help/index.jsp>

9.4.5 DB2 UDB for iSeries

In administering DB2 UDB for iSeries, the iSeries Navigator feature of iSeries Access for Windows can be used. It provides a GUI that allows you to manage DB2 files using TCP/IP connectivity for clients running a variety of Microsoft Windows operating systems. This product uses the client/server architecture. Make sure you install the 5722XE1 product on your iSeries too.

To order a copy of the iSeries Access for Windows product, follow this link:

<http://www-1.ibm.com/servers/eserver/iseries/access/>

Note: Ensure that the latest Service Packs are installed on the client with iSeries Access for Windows. The service packs can be downloaded from the same Web site.

Once you have installed and configured iSeries Access for Windows, you can open iSeries Navigator by clicking, for example, **Start** → **All Programs** → **IBM iSeries Access for Windows** → **iSeries Navigator**.

Figure 9-19 shows the sample window.

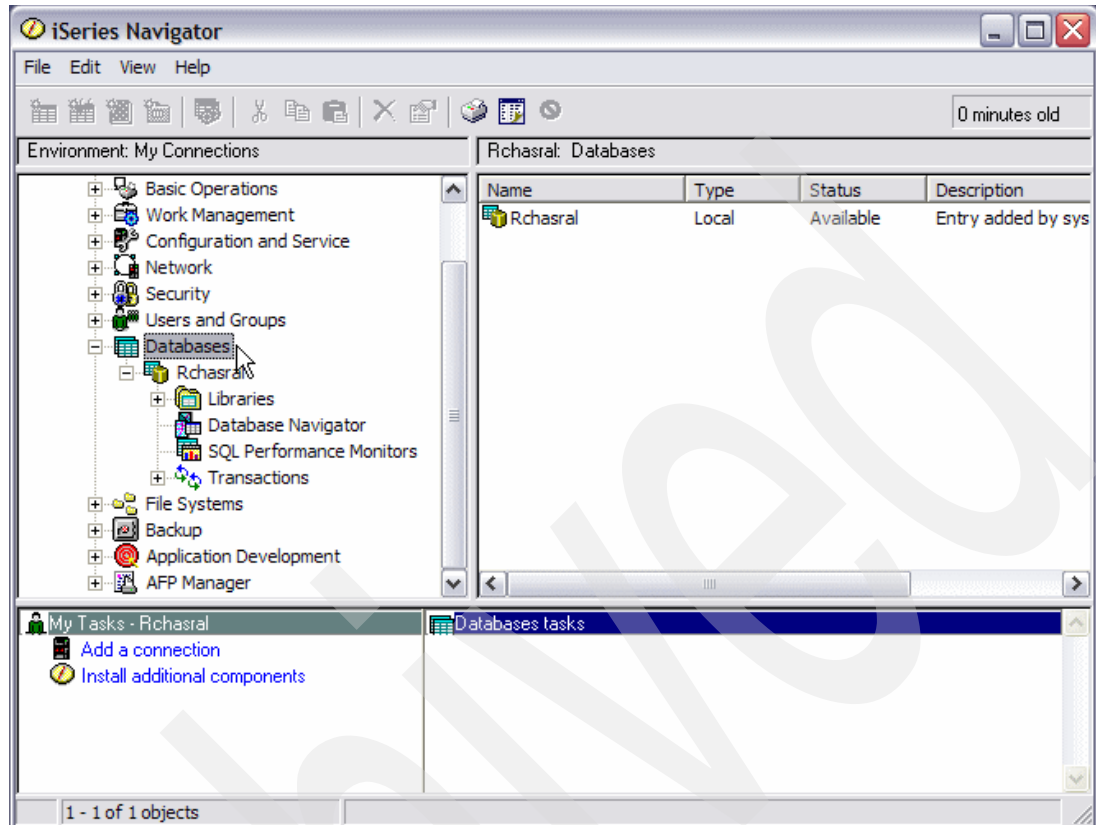


Figure 9-19 iSeries navigator

With iSeries navigator, you can perform DB2 administrative database tasks. Refer to the following URL for details about using the iSeries navigator:

<http://www-1.ibm.com/servers/eserver/iseries/access/caprod.htm>

9.4.6 Web Administration for iSeries

Web Administration for iSeries can be used to manage WebSphere Application Server - Express and IBM HTTP server. You can access this GUI by using these steps:

1. Start the *ADMIN instance of HTTP Server:
`STRTCPSVR SERVER(*HTTP) HTTPSVR(*ADMIN)`
2. Connect to your iSeries with the Web browser on port 2001 (default port).

This is the easiest interface to use for managing these two products on iSeries.

Archived



Migrating wrappers to Express Runtime V2.1

With the advent of Express Runtime V2.1, a need arises to migrate application and solution projects from Express Runtime's predecessor - Integrated Runtime V1.1. This chapter describes ways to do the migration.

The following topics are discussed:

- ▶ Moving projects and deployment package files
- ▶ Updating wrapper files
- ▶ Generating the solution

10.1 Moving projects and deployment package files

Migrating application and solution projects are fairly straightforward. The following scenarios can be used in migrating or moving projects to Express Runtime V2.1 or to another physical system:

- ▶ In moving projects to another physical system, they can be exported to a removable media, using **File** → **Export** → **File system**. The bin folder does not have to be included. However, be sure to include the .classpath and .project files (see Figure 10-1).

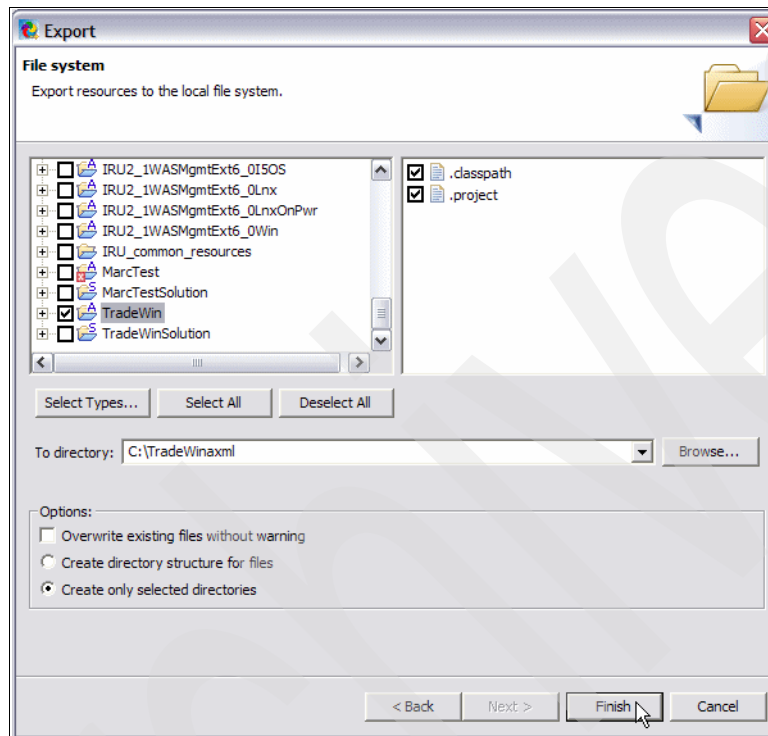


Figure 10-1 Migrating to another physical system

- ▶ For migrating projects from Integrated Runtime V1.1 to the Express Runtime V2.1 developer workspace, just follow these options: **File** → **Import** → **Existing project into workspace**. This makes the necessary updates to the imported wrappers - adding new XML tags and updating classpath information. The import wizard does not physically move or copy the imported files. If you prefer to have the imported project's files physically reside under the target workspace folder, ensure that the project to be imported is placed there prior to the import.
- ▶ If you are migrating an application that was included with Integrated Runtime V1.1, copy the application's deployment package from the installation Integrated Runtime V1.1 directory to the Express Runtime V2.1 directory.
 - Locate the deployment package JAR file in the Integrated Runtime V1.1 installation directory (*IR install dir* \SolutionEnabler\com\ibm\jsdt\webserver\tree). The deployment package filename contains the application ID and an abbreviation of the platform it is intended for (for example, "win" = Windows, "lnx" = Linux).

Note: *IR install dir* refers to C:\Program Files\IBM\Runtime.

- Copy the application's deployment package to the Express Runtime V2.1 installation directory (*ER install dir*\SolutionEnabler\com\ibm\jsdt\webserver\tree).

Note: *ER install dir* refers to C:\Program Files\IBM\Runtime21.

- Rename the copy of the deployment package in the Express Runtime V2.1 installation directory, changing the “.en.jar” suffix to “.xx.jar”

10.2 Updating wrapper files

After importing a project for migration, the wrappers need to be reviewed and updated. Follow these pointers in updating wrappers:

- ▶ For each project you want to migrate, open the solution or application wrapper in the appropriate wrapper editor.
- ▶ Review the new updates generated by the import process to ensure the correct values have been used and to add information to the optional attributes.
- ▶ Optionally remove deprecated tags using the Source tab in the wrapper editor to edit the XML code. The deprecated tags are listed in the Problems view (see Figure 10-2) in the developer. However, the wrapper does work in Express Runtime V2.1 if you leave the deprecated tags.

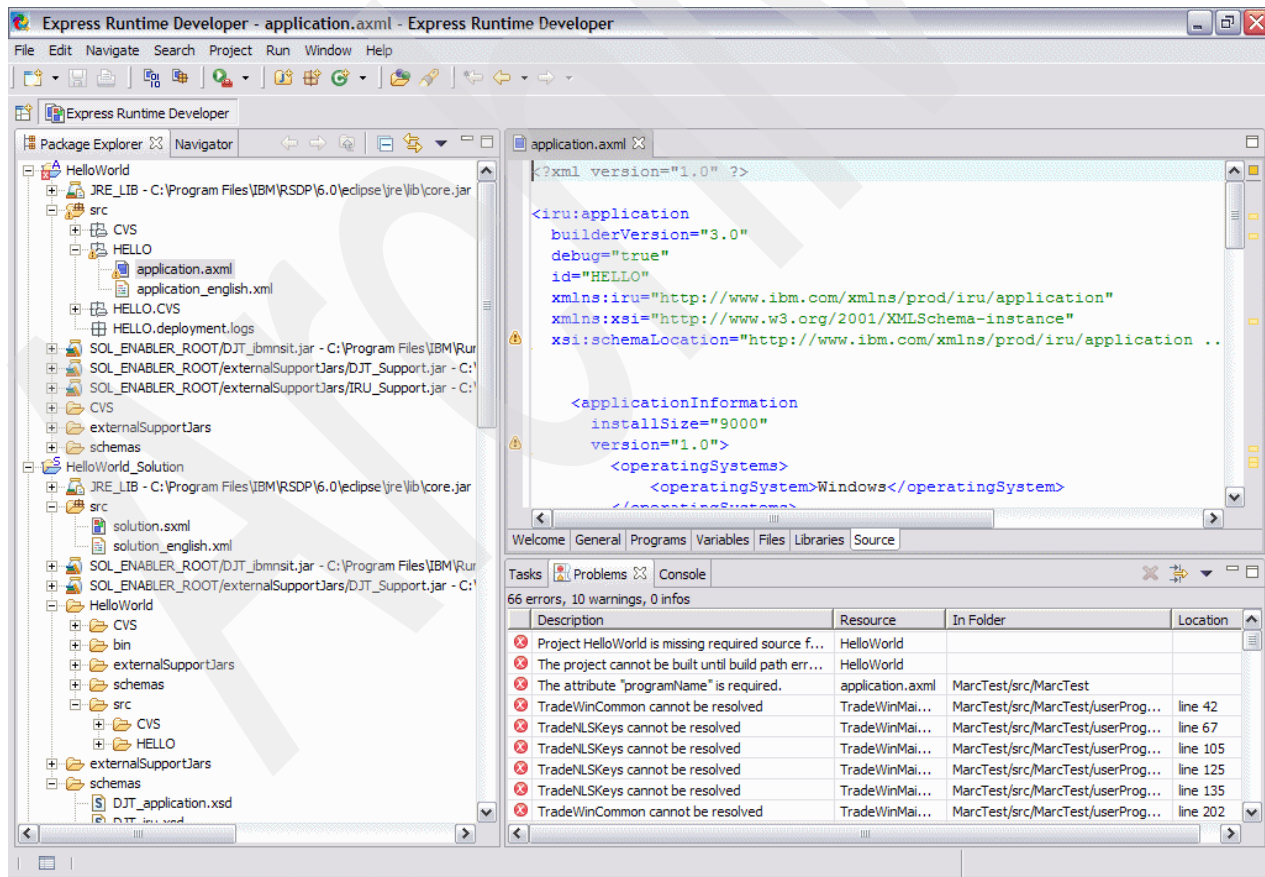


Figure 10-2 Problems view

10.2.1 Generating the solution

After migrating the wrapper files to Express Runtime V2.1, the project needs to be generated using Express Runtime Developer. This builds the binary files needed to deploy your solution with the new Express Runtime V2.1. See 6.2.6, “Building the Trade6 solution” on page 143 for more information.

Appendixes

This part of the book contains information related to the sample applications and solutions described in the book. You can find the source code here for most of the user programs and other files.

The following appendixes are included:

- ▶ Appendix A, “Source code for Trade6 user programs and script files on Windows” on page 257
- ▶ Appendix B, “Source code for Trade6 user programs and script files for Linux on POWER” on page 323
- ▶ Appendix C, “Source code for Flight400 user programs and script files on OS/400” on page 395
- ▶ Appendix D, “Additional material” on page 435

Archived

Source code for Trade6 user programs and script files on Windows

This appendix provides the source codes used in the Trade6 solution example. The following programs and files are included:

- ▶ application.xml
- ▶ solution.xml
- ▶ TradeWinMain.java
- ▶ TradeWinPDC.java
- ▶ TradeWinCommon.java
- ▶ TradeNLSKeys.java
- ▶ TradeMessagesNLS.java
- ▶ CheckAppInstall.jacl
- ▶ WebSphereConfigProcs.jacl
- ▶ WebSphereScript.jacl
- ▶ Table.ddl
- ▶ DB2Script.bat
- ▶ SetupProcs.jacl
- ▶ Trade.prop file

The application.xml file

Example A-1 shows the source of the application.xml file.

Example: A-1 Source of application.xml

```
<?xml version="1.0" ?>

<iru:application
  id="TradeWin"
  xmlns:iru="http://www.ibm.com/xmlns/prod/iru/application"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/xmlns/prod/iru/application IRU_application.xsd">

  <applicationInformation
    installTime="20"
    version="1.0">
    <name>TradeWin</name>
    <operatingSystems>
      <operatingSystem>Windows</operatingSystem>
    </operatingSystems>
    <providerName>IBM</providerName>
  </applicationInformation>

  <translationLanguages default="english">
    <language>english</language>
  </translationLanguages>

  <fileLists>
    <fileList id="softwareimagefiles">
      <file>trade.ear</file>
    </fileList>
    <fileList
      id="userprogramfiles"
      userPrograms="true">
      <file>com/trade/TradeMessagesNLS.class</file>
      <file>com/trade/TradeMessagesNLS_en.class</file>
      <file>com/trade/TradeNLSKeys.class</file>
      <file>com/trade/TradeWinCommon.class</file>
      <file>com/trade/TradeWinMain.class</file>
      <file>com/trade/TradeWinPDC.class</file>
      <file>Trade_ScriptsDir/CheckAppInstall.jacl</file>
      <file>Trade_ScriptsDir/DB2Script.bat</file>
      <file>Trade_ScriptsDir/SetupProcs.jacl</file>
      <file>Trade_ScriptsDir/Table.ddl</file>
      <file>Trade_ScriptsDir/WebSphereConfigProcs.jacl</file>
      <file>Trade_ScriptsDir/WebSphereScript.jacl</file>
    </fileList>
  </fileLists>

  <preDeploymentChecker
    logFile="TradeWinPDC.log"
    programName="com.trade.TradeWinPDC"
    responseFile="Trade.prop"
    successType="returnCode"
    type="java">
```

```

    <arguments>
      <argument responseFile="true" />
    </arguments>
  </preDeploymentChecker>

```

```

<mainProgram
  logFile="TradeWinMain.log"
  programName="com.trade.TradeWinMain"
  responseFile="Trade.prop"
  successType="returnCode"
  type="java">
    <arguments>
      <argument responseFile="true" />
    </arguments>
  </mainProgram>

```

```

<variables>
  <stringVariable
    maxLength="30"
    minLength="2"
    name="DB2UserId"
    required="true">
    <labelText>DB2 Administrator UserID</labelText>
    <propertiesAssociations>
      <propertiesAssociation keyword="DB2UserId" />
    </propertiesAssociations>
    <inputValidation>
      <invalid>
        <prefixes>
          <prefix ignoreCase="true">IBM</prefix>
          <prefix ignoreCase="true">SQL</prefix>
          <prefix ignoreCase="true">SYS</prefix>
          <prefix ignoreCase="true">_</prefix>
        </prefixes>
        <values>
          <value ignoreCase="true">ADMINS</value>
          <value ignoreCase="true">GUESTS</value>
          <value ignoreCase="true">USERS</value>
          <value ignoreCase="true">PUBLIC</value>
          <value ignoreCase="true">LOCAL</value>
        </values>
      </invalid>
      <valid>
        <characters ignoreCase="true">@#$_abcdefghijklmnopqrstuvwxyz0123456789</characters>
      </valid>
    </inputValidation>
    <helpText>The Administrator id used to connect to DB2.</helpText>
  </stringVariable>
  <passwordVariable
    maxLength="127"
    minLength="6"
    name="DB2UserPassword"
    required="true">
    <labelText>DB2 Administrator Password</labelText>
    <propertiesAssociations>
      <propertiesAssociation keyword="DB2UserPassword" />
    </propertiesAssociations>
    <inputValidation>

```

```

        <valid>
            <characters ignoreCase="true">@#$_abcdefghijklmnopqrstuvwxyz0123456789</characters>
        </valid>
    </inputValidation>
    <helpText>The Password used with the id specified to connect to DB2.</helpText>
</passwordVariable>
</variables>

```

```
</iru:application>
```

The solution.xml file

Example A-2 shows the source of the solution.xml file.

Example: A-2 Source of solution.xml

```

<?xml version="1.0" ?>

<iru:solution
  id="TradeWinSolution"
  version="6.0"
  xmlns:iru="http://www.ibm.com/xmlns/prod/iru/solution"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/xmlns/prod/iru/solution IRU_solution.xsd">

  <solutionInformation>
    <title>TradeWinSolution</title>
    <welcomeScreenTitle>Trade V6 Solution</welcomeScreenTitle>
    <welcomeScreenText>The solution to deploy all middleware, the Trade application, and Console for
Express Runtime.</welcomeScreenText>
    <aboutScreenText>&lt;![CDATA[&lt;html&gt;&lt;p align="left"&gt;&lt;p
style="margin-left:25px&gt;&lt;font size="2&quot;&gt;&lt;BR&gt;Trade
Solution&lt;BR&gt;&lt;/font&gt;&lt;/style&gt;&lt;/align&gt;&lt;/html&gt;]]&gt;</aboutScreenText>
    <taskGroupSelectionPrompt>Select which tasks you would like to deploy</taskGroupSelectionPrompt>
  </solutionInformation>

  <translationLanguages default="english">
    <language>english</language>
  </translationLanguages>

  <tasks>
    <taskGroup>
      <taskGroupTitle>Trade Application</taskGroupTitle>
      <taskGroupPrompt>Select the platforms where the middleware and Trade application will be
installed</taskGroupPrompt>
      <installTask
        isOptional="true"
        operatingSystem="Windows">
        <description>Trade Application and Middleware for Windows</description>
        <applications>
          <application fileName="IRU2_1DB2Express8_2Win_win.ser">
            <variables>
              <variable
                hidden="false"

```

```

        id="username"
        sharedAs="DB2AdminUsernameWin" />
    <variable
        hidden="false"
        id="password"
        sharedAs="DB2AdminPasswordWin" />
    </variables>
</application>
<application fileName="IRU2_1DB2MgmtExt8_2Win_win.ser" />
<application fileName="IRU2_1WASExpress6_0Win_win.ser">
    <variables>
        <variable
            hidden="false"
            id="installLocation"
            sharedAs="WASInstallDirWin" />
        </variables>
    </application>
<application fileName="IRU2_1WASMgmtExt6_0Win_win.ser" />
<application fileName="IRU2_1IHS6_0Win_win.ser">
    <variables>
        <variable
            hidden="false"
            id="installDir"
            sharedAs="IHSInstallDirWin" />
        </variables>
    </application>
<application fileName="IRU2_1WASExpressHttpPlugin6_0Win_win.ser">
    <variables>
        <variable
            hidden="false"
            id="ihsInstallLocation"
            sharedAs="IHSInstallDirWin" />
        <variable
            hidden="false"
            id="wasInstallLocation"
            sharedAs="WASInstallDirWin" />
        <variable
            hidden="true"
            id="appServer" />
        </variables>
    </application>
<application fileName="IRU2_1IHSMgmtExt6_0Win_win.ser">
    <variables>
        <variable
            hidden="false"
            id="httpServerDir"
            sharedAs="IHSInstallDirWin" />
        </variables>
    </application>
<application fileName="IRU2_1ConsoleWin_win.ser" />
<application fileName="TradeWin_win.ser">
    <variables>
        <variable
            hidden="false"
            id="DB2UserId"
            sharedAs="DB2AdminUsernameWin" />
        <variable
            hidden="false"
            id="DB2UserPassword"
            sharedAs="DB2AdminPasswordWin" />
    </variables>
</application>

```

```

        </variables>
    </application>
</applications>
</installTask>
</taskGroup>
</tasks>

```

```

<variables>
  <sharedVariable
    maxLength="14"
    minLength="1"
    name="DB2AdminPasswordWin"
    required="true">
      <inputValidation>
        <valid>
          <characters ignoreCase="true">@#$_abcdefghijklmnopqrstuvwxyz0123456789</characters>
        </valid>
      </inputValidation>
    </sharedVariable>
  <sharedVariable
    maxLength="30"
    minLength="1"
    name="DB2AdminUsernameWin"
    required="true">
      <inputValidation>
        <invalid>
          <prefixes>
            <prefix ignoreCase="true">IBM</prefix>
            <prefix ignoreCase="true">SQL</prefix>
            <prefix ignoreCase="true">SYS</prefix>
            <prefix>_</prefix>
            <prefix>0</prefix>
            <prefix>1</prefix>
            <prefix>2</prefix>
            <prefix>3</prefix>
            <prefix>4</prefix>
            <prefix>5</prefix>
            <prefix>6</prefix>
            <prefix>7</prefix>
            <prefix>8</prefix>
            <prefix>9</prefix>
          </prefixes>
          <values>
            <value ignoreCase="true">ADMINS</value>
            <value ignoreCase="true">GUESTS</value>
            <value ignoreCase="true">USERS</value>
            <value ignoreCase="true">PUBLIC</value>
            <value ignoreCase="true">LOCAL</value>
          </values>
        </invalid>
        <valid>
          <characters ignoreCase="true">@#$_abcdefghijklmnopqrstuvwxyz0123456789</characters>
        </valid>
      </inputValidation>
      <defaultData>db2admin</defaultData>
    </sharedVariable>
  <sharedVariable
    maxLength="130"
    name="IHSInstallDirWin"

```



```

required="true">
  <inputValidation>
    <valid>
      <prefixes>
        <prefix ignoreCase="true">A:\</prefix>
        <prefix ignoreCase="true">B:\</prefix>
        <prefix ignoreCase="true">C:\</prefix>
        <prefix ignoreCase="true">D:\</prefix>
        <prefix ignoreCase="true">E:\</prefix>
        <prefix ignoreCase="true">F:\</prefix>
        <prefix ignoreCase="true">G:\</prefix>
        <prefix ignoreCase="true">H:\</prefix>
        <prefix ignoreCase="true">I:\</prefix>
        <prefix ignoreCase="true">J:\</prefix>
        <prefix ignoreCase="true">K:\</prefix>
        <prefix ignoreCase="true">L:\</prefix>
        <prefix ignoreCase="true">M:\</prefix>
        <prefix ignoreCase="true">N:\</prefix>
        <prefix ignoreCase="true">O:\</prefix>
        <prefix ignoreCase="true">P:\</prefix>
        <prefix ignoreCase="true">Q:\</prefix>
        <prefix ignoreCase="true">R:\</prefix>
        <prefix ignoreCase="true">S:\</prefix>
        <prefix ignoreCase="true">T:\</prefix>
        <prefix ignoreCase="true">U:\</prefix>
        <prefix ignoreCase="true">V:\</prefix>
        <prefix ignoreCase="true">W:\</prefix>
        <prefix ignoreCase="true">X:\</prefix>
        <prefix ignoreCase="true">Y:\</prefix>
        <prefix ignoreCase="true">Z:\</prefix>
      </prefixes>
    </valid>
    <invalid>
      <substrings>
        <substring>\\</substring>
        <substring>//</substring>
      </substrings>
      <characters>&lt;&gt;*&quot;/|</characters>
    </invalid>
  </inputValidation>
  <defaultData>C:\Program Files\IBM HTTP Server</defaultData>
</sharedVariable>
<sharedVariable
  maxLength="65"
  name="WASInstallDirWin"
  required="true">
  <defaultData>C:\Program files\IBM\WebSphere\AppServer</defaultData>
  <inputValidation>
    <invalid>
      <characters>*&quot;/|&lt;&gt;</characters>
      <substrings>
        <substring>\\</substring>
        <substring>//</substring>
      </substrings>
    </invalid>
    <valid>
      <prefixes>
        <prefix ignoreCase="true">A:\</prefix>
        <prefix ignoreCase="true">B:\</prefix>
        <prefix ignoreCase="true">C:\</prefix>

```

```

    <prefix ignoreCase="true">D:\</prefix>
    <prefix ignoreCase="true">E:\</prefix>
    <prefix ignoreCase="true">F:\</prefix>
    <prefix ignoreCase="true">G:\</prefix>
    <prefix ignoreCase="true">H:\</prefix>
    <prefix ignoreCase="true">I:\</prefix>
    <prefix ignoreCase="true">J:\</prefix>
    <prefix ignoreCase="true">K:\</prefix>
    <prefix ignoreCase="true">L:\</prefix>
    <prefix ignoreCase="true">M:\</prefix>
    <prefix ignoreCase="true">N:\</prefix>
    <prefix ignoreCase="true">O:\</prefix>
    <prefix ignoreCase="true">P:\</prefix>
    <prefix ignoreCase="true">Q:\</prefix>
    <prefix ignoreCase="true">R:\</prefix>
    <prefix ignoreCase="true">S:\</prefix>
    <prefix ignoreCase="true">T:\</prefix>
    <prefix ignoreCase="true">U:\</prefix>
    <prefix ignoreCase="true">V:\</prefix>
    <prefix ignoreCase="true">W:\</prefix>
    <prefix ignoreCase="true">X:\</prefix>
    <prefix ignoreCase="true">Y:\</prefix>
    <prefix ignoreCase="true">Z:\</prefix>
  </prefixes>
</valid>
</inputValidation>
</sharedVariable>
</variables>

```

```
</iru:solution>
```

The TradeWinMain.java program

Example A-3 shows the complete listing of the TradeWinMain class.

Example: A-3 Source of TradeWinMain

```

package com.trade;

import com.ibm.jsdt.support.*;

/**
 * @author IBM_USER
 *
 */
public class TradeWinMain extends SupportWindowsBase {

    private SupportWindowsHelper ivHelper = null;

    private String ivCommandResult = null;

    private String ivDB2UserId = null;
    private String ivDB2PassWord = null;
    private String ivWasVersion = null;
    private String ivWasInstallDir = null;
    private String ivWasProfileName = null;
    private String ivInstallableAppsDir = null;

```

```

private String ivWasProfileBinDir = null;
private String ivDB2InstallDir = null;
private String ivAppFile = null;
private String ivDB2Script = null;
private String ivScriptsDir = null;
private String ivDbName = null;
private String ivWasScript = null;
private String ivIhsServerName = null;
private String ivServerName = null;

/**
 * @param args
 */
public TradeWinMain(String[] args) {

    ivHelper = (SupportWindowsHelper) getHelper();
    setJarFile(ivHelper.getProductInstallingId(this));
    setMainResources(TradeWinCommon.SAMPLE_MESSAGES);
    // The properties file is in the same format as a response file
    setResponseFileName((1 == args.length) ? args[0] : null);

}

public static void main(String[] args) {
    System.out.println("Starting");
    TradeWinMain tradeMain = new TradeWinMain(args);
    int rc = tradeMain.install();
    System.out.println("Ending with rc= " + rc);
    System.exit(rc);
}

/**
 * @return
 */
private int install() {
    int rc = SUCCESS;

    // get the values set in the properties file, properties file is
    // mandatory
    if (!(getResponseFileName() == null || getResponseFileName().trim().equals("")) ) {
        rc = getProperties();
    } else {
        setMessage(getResourceString(TradeNLSKeys.NO_PROPERTIES_FILE));
        ivHelper.log(this);
        rc = FAILURE;
    }

    // Determine if WAS is installed and get the WAS directories
    if (rc == SUCCESS) { rc = determineWasDir(); }

    // Determine if DB2 is installed
    if (rc == SUCCESS) { rc = determineDB2Dir(); }

    // Set values in the properties file
    if (rc == SUCCESS) { setProperties(); }

    if (rc == SUCCESS) {
        // copy the application EAR or WAR file into the "installable apps" directory
        rc = copyApp();
    }
}

```

```

    if (rc == SUCCESS) {
        // run the DB/2 Script
        rc = runDB2Script();
    }
    if (rc == SUCCESS) {
        // Run the WAS Script
        rc = runWebSphereScript();
    }
    if (rc == SUCCESS) {
        // Restart WAS
        rc = restartWAS();
    }

    // Generate the HTTP Plugin
    if (rc == SUCCESS) {
        rc = generateHTTPPlugin();
    }

    // Restart the HTTP Server so the plugin will be enabled
    if (rc == SUCCESS) {
        // if IHS specified in the application properties file -- restart it
        if (getIhsServerName() != null && !getIhsServerName().trim().equals("")) {
            rc = restartHTTPServer();
        } else {
            setMessage(getResourceString(TradeNLSKeys.NO_HTTPSERVER));
            ivHelper.log(this);
        }
    }

    return rc;
}

/**
 * @return
 */
private int restartWAS() {
    int rc = SUCCESS;

    // stop the server, log an error but continue if an error occurs
    String command = "cmd /C " + getWasProfileBinDir() + "stopServer " + getServerName();
    int ignoredRC = invokeCommand(command);
    if (ignoredRC != SUCCESS) {
        // log an error if not successful
        setMessage(getResourceString(TradeNLSKeys.STOP_HTTPSERVER_FAIL));
        ivHelper.log(this);
    }

    // start the server
    command = "cmd /C " + getWasProfileBinDir() + "startServer " + getServerName();
    rc = invokeCommand(command);

    if (rc != SUCCESS) {
        // log an error if not successful
        setMessage(getResourceString(TradeNLSKeys.START_HTTPSERVER_FAIL));
        ivHelper.log(this);
    }
}

```

```

        return rc;
    }

    /**
     * @return
     */
    private int restartHTTPServer() {
        int rc = SUCCESS;

        // use the net command to stop the server, log an error but continue if an error occurs
        String command = "cmd /C net stop " + getIhsServerName();
        int ignoredRC = invokeCommand(command);
        if (ignoredRC != SUCCESS) {
            // log an error if not successful
            setMessage(getResourceString(TradeNLSKeys.STOP_HTTPSERVER_FAIL));
            ivHelper.log(this);
        }

        // use the net command to start the server
        command = "cmd /C net start " + getIhsServerName();
        rc = invokeCommand(command);

        if (rc != SUCCESS) {
            // log an error if not successful
            setMessage(getResourceString(TradeNLSKeys.START_HTTPSERVER_FAIL));
            ivHelper.log(this);
        }

        return rc;
    }

    /**
     * @return
     */
    private int generateHTTPPlugin() {
        int rc = SUCCESS;

        // Get the Windows short path for the WAS bin directory
        setPath(getWasProfileBinDir());
        String shortWasProfileBinDir = ivHelper.getWindowsShortPath(this);

        // Use the WAS GenPluginCfg to generate the HTTP plug-in config file
        String command = "cmd /C " + shortWasProfileBinDir + "GenPluginCfg.bat";
        rc = invokeCommand(command);

        return rc;
    }

    /**
     * @return
     */
    private int runWebSphereScript() {
        int rc = SUCCESS;

        // Get the WAS script to be run
        // The script name is specified in the application properties file - DOCMGMT.WASscript
        String script = getWasScript();
        if ((script != null) && !script.trim().equals("")) {
            String unpackedDir = ivHelper.getUnpackedDir(this);

```

```

// Get the Windows short path for the WAS bin directory
setPath(getWasProfileBinDir());
String shortWasProfileBinDir = ivHelper.getWindowsShortPath(this);

// Get the Windows short path for the response file
//setPath(getResponseFileName());
//String shortResponseFileName = ivHelper.getWindowsShortPath(this);

// The directory where the scripts are located are in the
// unpackedDir. A subdirectory may have been specified for these in the
// application.xml. The application properties file has this in DOCMGMT.scriptsDir
String fullScriptsDir = ivHelper.getUnpackedDir(this).replace('\\', '/')
+ getScriptsDir() + "/";

// Run the WebSphere Admin application (wsadmin)
// The -f option indicates a jacl script is the next arg
// After the script name, the args for the jacl script are:
// - directory where the scripts are located
// - the fully qualified properties file name
// - the DB2 administrator userID
// - the DB2 administrator password
String command = "cmd /C " +
    shortWasProfileBinDir +
    "wsadmin -f " +
    fullScriptsDir + getWasScript() + // Main jacl script
    " " + fullScriptsDir + // the scripts directory
    " " + getResponseFileName().replace('\\', '/'); // and the properties file
rc = invokeCommand(command, command);

// Search the command result for error messages
// If an error string is found then set rc=FAILURE
if (TradeWinCommon.messageExists(getCommandResult(), TradeWinCommon.WAS_WSADMIN_ERRORS)) {
    rc = FAILURE;
}
} else {
    setMessage(getResourceString(TradeNLSKeys.NO_WAS_SCRIPT));
    ivHelper.log(this);
}

return rc; }

/**
 * @return
 */
private int runDB2Script() {
    int rc = SUCCESS;

    // Get the DB2 script to be run
    // The script name is specified in the application properties file - DOCMGMT.DB2script
    String script = getDB2Script();

    if ((script != null) && !script.trim().equals("")) {
        // The directory where the scripts are located are in the
        // unpackedDir. A subdirectory may have been specified for these in the
        // application.xml. The application properties file has this in DOCMGMT.scriptsDir
        String fullScriptsDir = ivHelper.getUnpackedDir(this).replace('\\', '/') + getScriptsDir() +
"/";

        // Check if the script is a batch or cmd file

```

```

String lowerCaseScript = script.toLowerCase();
if (lowerCaseScript.endsWith(".bat") || lowerCaseScript.endsWith(".cmd")) {
    // Run batch file
    // db2cmd command options:
    // -c is to execute the command and then terminate
    // -i is to share same console (window)
    // -w is to wait until the cmd.exe process ends
    // The name of the script is then followed by the args used by the script
    // - the database name
    // - the full path to the script dir
    // - the full path to the documents dir
    // - the db2 user
    // - the db2 password

    String command = "db2cmd -c -i -w " +
        fullScriptsDir + script +
        " " + getDbName() +
        " " + fullScriptsDir +
        " " + getDB2UserId() +
        " " + getDB2PassWord();
        // Script to run
        // database name
        // Scripts directory
        // DB2 userid
        // DB2 password

    rc = invokeCommand(command);
} else {
    // Assume this is a file with DB2 commands
    // db2cmd command options:
    // -c is to execute the command and then terminate
    // -i is to share same console (window)
    // -w is to wait until the cmd.exe process ends
    // db2 command options:
    // -c is to auto-commit
    // -w is to display SQL warning messages
    // -o is to display output
    // -f is to read the db2 commands from a file
    // the name of the script is passed with no additional args
    String command = "db2cmd -c -i -w db2 -c -w -o -f \"" + fullScriptsDir + script + "\"";
    rc = invokeCommand(command);
    ivHelper.logAppendFile(this);
}
} else {
    setMessage(getResourceString(TradeNLSKeys.NO_DB2_SCRIPT));
    ivHelper.log(this);
}
return rc; }

/**
 * @return
 */
private int copyApp() {
    int rc = FAILURE;

    // Copy the ear or war file to the WAS installableApps directory
    setSource(ivHelper.getUnpackedDir(this) + getAppFile());
    setTarget(getInstallableAppsDir());

    if (ivHelper.fileCopy(this)) {
        setMessage(getResourceString(TradeNLSKeys.COPYFILE_SUCCESS, getSource() + "," + getTarget()));
        rc = SUCCESS;
    } else {
        setMessage(getResourceString(TradeNLSKeys.COPYFILE_FAIL, getSource() + "," + getTarget()));
        rc = FAILURE;
    }
}

```

```

    }

    ivHelper.log(this);
    return rc;
}

/**
 *
 */
private void setProperties() {
    // set the JDBC Provider classpath in the properties file
    setKey(TradeWinCommon.JDBC_CLASSPATH_KEY);
    // the JDBC classpath is based on the DB2 install directory
    setKeyValue(getDB2InstallDir() + "java/db2jcc.jar;" + getDB2InstallDir() +
"java/db2jcc_license_cu.jar;" + getDB2InstallDir() + "java/db2java.zip");
    ivHelper.setResponseFileValue(this);

    // use the support framework to convert the installableApps directory to a short path
    setPath(getInstallableAppsDir());
    String shortInstallableAppsDir = ivHelper.getWindowsShortPath(this);
    // set the WAS installable apps directory in the properties file
    setKey(TradeWinCommon.WAS_INSTALLABLE_APPS_DIR_KEY);
    setKeyValue(shortInstallableAppsDir);
    ivHelper.setResponseFileValue(this);

    // include the contents of the updated properties file in the log
    ivHelper.logNewLine(this);
    setFileName(getResponseFileName());
    ivHelper.logAppendFile(this);
    ivHelper.logNewLine(this);

    // set the db2 administrator user id and password
    setKey(TradeWinCommon.DB2_ADMIN_ID_KEY);
    setKeyValue(getDB2UserId());
    ivHelper.setResponseFileValue(this);
    setKey(TradeWinCommon.DB2_ADMIN_PW_KEY);
    setKeyValue(getDB2PassWord());
    ivHelper.setResponseFileValue(this);
}

/**
 * @return
 */
private int determineWasDir() {
    int rc = SUCCESS;

    setWasInstallDir(TradeWinCommon.determineWasDir(this, ivHelper, getWasVersion()));
    if (getWasInstallDir() == null) {
        setMessage(getResourceString(TradeNLSKeys.WAS_NOT_INSTALLED));
        ivHelper.log(this);
        rc = FAILURE;
    } else {
        // Set the WAS profile bin dir
        setWasProfileBinDir(getWasInstallDir() +
"profiles/" +
getWasProfileName() + "/bin/");
        setInstallableAppsDir(getWasInstallDir() + "installableApps/"); // set the installableApps
directory
    }
}

```



```

        return rc;
    }

    /**
     * @return
     */
    private int determineDB2Dir() {
        int rc = SUCCESS;
        setDB2InstallDir(TradeWinCommon.determineDB2Dir(this, ivHelper));
        if (getDB2InstallDir() == null) {
            setMessage(getResourceString(TradeNLSKeys.DB2_NOT_INSTALLED));
            ivHelper.log(this);
            rc = FAILURE;
        }
        return rc;
    }

    /**
     * @return
     */
    private int getProperties() {
        // This is where we get all the properties needed to do this install
        // They can come from a variety of places
        // The first place is the response file that was passed in to the Main program (Trade.prop)
        // This can be accessed using the getResponseFileValue method and a key:
        //
        // setKey(SOME_KEY_VALUE);
        // ivHelper.getResponseFileValue(this);
        //
        //
        // Trade.props contains pre-defined values. However, these can be overwritten by the deployer
        // at deployment time as follows:
        // application.xml contains variables. These are values that the deployer will provide at deploy
time.
        // Any variable defined in application.xml can also define an association with a response file.
        // Defining a association between a variable and a response file causes the Deployment Wizard
        // to automatically update the response file with whatever value the deployer enters for the
variable.
        // This means that the above API will actually access values from the updated response file.
        // Note that CID, ISMP and ISS style response files can be updated in this way.
        //
        // In some cases this technique will not be ideal. This is because the Main program will probably
want to pass
        // the response file into a silent installer script/program at some point. Often the
installer/script will be strict
        // about the syntax and contents of the response file it expects. This means that the content of
the response file
        // is not completely flexible. For example you may wish to pass a value to the Main program that is
not needed
        // by the installer and worse still will cause the installer to fail if it is present.
        //
        // In such a scenario there is an alternative way of receiving input from the deployer without
inserting it into
        // the properties file. Any variable can be inserted into another file called IBMNSI.properties
that exists only
        // during the execution of the Deployment Wizard. The values in this file can then be accessed
using an alternative API:
        //

```

```

        // setVariableName(SOME_VARIABLE_NAME);
        // ivHelper.getIbmNsiPropValue(this);
        //
        // For a variable to be accessible in the IBMNSI.properties file it must have an association
created. If the association
        // is to a response file then the name of the variable in the IBMNSI file will be the same as the
variable name.
        // If you want you can specify a 'property association' (as well as the response file association)
which allows you to alter
        // the name that is used in the IBMNSI.properties file.
        //
        //
*****

```

```

        int rc = SUCCESS;
        // Get the properties from the support framework Nsi Properties file
        setVariableName("DB2UserId");
        setDB2UserId(ivHelper.getIbmNsiPropValue(this));
        setVariableName("DB2UserPassword");
        setDB2PassWord(ivHelper.getIbmNsiPropValue(this));

        // Get the values specified in the application properties file
        // The application properties file name was set in the constructor, this file is
        // in the same format as a response file therefore the support framework
        // getResponseFileValue and setResponseFileValue methods can be used
        setKey(TradeWinCommon.WAS_VERSION_KEY); // WAS version is required to get the install path and bin
path
        setWasVersion(ivHelper.getResponseFileValue(this));
        if (getWasVersion() == null || getWasVersion().trim().equals("")) {
            setMessage(getResourceString(TradeNLSKeys.NO_PROPERTY, TradeWinCommon.WAS_VERSION_KEY));
            ivHelper.log(this);
            rc = FAILURE;
        }

        setKey(TradeWinCommon.WAS_PROFILE_KEY); // WAS profile name
        setWasProfileName(ivHelper.getResponseFileValue(this));
        if (getWasProfileName() == null || getWasProfileName().trim().equals("")) {
            setMessage(getResourceString(TradeNLSKeys.NO_PROPERTY, TradeWinCommon.WAS_PROFILE_KEY));
            ivHelper.log(this);
            rc = FAILURE;
        }

        setKey(TradeWinCommon.APP_FILE_KEY); // the ear or war file to be installed
        setAppFile(ivHelper.getResponseFileValue(this));
        setKey(TradeWinCommon.DB2_SCRIPT_KEY); // DB2 script
        setDB2Script(ivHelper.getResponseFileValue(this));
        setKey(TradeWinCommon.SCRIPTS_DIR_KEY); // the directory within userPrograms where the scripts are
located
        setScriptsDir(ivHelper.getResponseFileValue(this));
        setKey(TradeWinCommon.DB_NAME_KEY); // the name of the database
        setDbName(ivHelper.getResponseFileValue(this));
        setKey(TradeWinCommon.WAS_SCRIPT_KEY); // WAS script
        setWasScript(ivHelper.getResponseFileValue(this));
        setKey(TradeWinCommon.IHS_SERVER_NAME_KEY); // the name of the IHS server
        setIhsServerName(ivHelper.getResponseFileValue(this));
        setKey(TradeWinCommon.SERVER_NAME_KEY); // server name to use for the startWAS command
        setServerName(ivHelper.getResponseFileValue(this));

```

```

        return rc;
    }

/**
 * Invoke the command
 *   parm#0 = command
 */
private int invokeCommand(String command)
{
    return(invokeCommand(command, null));
}

/**
 * Invoke the command, commandLog is the text to be logged
 *   parm#0 = command
 *   parm#1 = command text to be logged
 *
 * This will set the command result in the instance
 * variable ivCommandResult in case the calling routine
 * needs to analyze the result
 *
 * The command and the result will both be logged here
 */
private int invokeCommand(String command, String commandLog)
{
    int rc = SUCCESS;

    try {
        setCommandResult(TradeWinCommon.invokeCommand (this, ivHelper, command, commandLog));
    } catch (Exception e) {
        rc = FAILURE;
        setMessage(getResourceString(TradeNLSKeys.CMD_EXCEPTION, e.toString()));
        ivHelper.log(this);
    }
    return rc;
}

private String getDB2PassWord() {
    return ivDB2PassWord;
}
private void setDB2PassWord(String ivDB2PassWord) {
    this.ivDB2PassWord = ivDB2PassWord;
}
private String getDB2UserId() {
    return ivDB2UserId;
}
private void setDB2UserId(String ivDB2UserId) {
    this.ivDB2UserId = ivDB2UserId;
}
private String getWasVersion() {
    return ivWasVersion;
}
}

```

```

private void setWasVersion(String ivWasVersion) {
    this.ivWasVersion = ivWasVersion;
}
private String getWasInstallDir() {
    return ivWasInstallDir;
}
private void setWasInstallDir(String ivWasInstallDir) {
    this.ivWasInstallDir = ivWasInstallDir;
}
private String getInstallableAppsDir() {
    return ivInstallableAppsDir;
}
private void setInstallableAppsDir(String ivInstallableAppsDir) {
    this.ivInstallableAppsDir = ivInstallableAppsDir;
}
private String getWasProfileName() {
    return ivWasProfileName;
}
private void setWasProfileName(String ivWasProfileName) {
    this.ivWasProfileName = ivWasProfileName;
}
private String getWasProfileBinDir() {
    return ivWasProfileBinDir;
}
private void setWasProfileBinDir(String ivWasProfileBinDir) {
    this.ivWasProfileBinDir = ivWasProfileBinDir;
}
private String getDB2InstallDir() {
    return ivDB2InstallDir;
}
private void setDB2InstallDir(String ivDB2InstallDir) {
    this.ivDB2InstallDir = ivDB2InstallDir;
}
private String getAppFile() {
    return ivAppFile;
}
private void setAppFile(String ivAppFile) {
    this.ivAppFile = ivAppFile;
}
private String getDB2Script() {
    return ivDB2Script;
}
private void setDB2Script(String ivDB2Script) {
    this.ivDB2Script = ivDB2Script;
}
private String getScriptsDir() {
    return ivScriptsDir;
}
private void setScriptsDir(String ivScriptsDir) {
    this.ivScriptsDir = ivScriptsDir;
}
private String getDbName() {
    return ivDbName;
}
private void setDbName(String ivDbName) {
    this.ivDbName = ivDbName;
}
private String getCommandResult() {
    return ivCommandResult;
}
}

```

```

private void setCommandResult(String ivCommandResult) {
    this.ivCommandResult = ivCommandResult;
}
private String getWasScript() {
    return ivWasScript;
}
private void setWasScript(String ivWasScript) {
    this.ivWasScript = ivWasScript;
}
private String getIhsServerName() {
    return ivIhsServerName;
}
private void setIhsServerName(String ivIhsServerName) {
    this.ivIhsServerName = ivIhsServerName;
}
private String getServerName() {
    return ivServerName;
}
private void setServerName(String ivServerName) {
    this.ivServerName = ivServerName;
}
}

```

The TradeWinPDC.java program

Example A-4 shows the source code for the TradeWinPDC class.

Example: A-4 Source of TradeWinPDC

```

package com.trade;

/**
 * @(#)TradeWinPDC.java
 */

import com.ibm.jsdt.support.SupportWindowsBase;
import com.ibm.jsdt.support.SupportWindowsHelper;

public class TradeWinPDC extends SupportWindowsBase {
    private String ivCommandResult = null;
    private String ivWASInstallDir = null;
    private String ivWASProfileBinDir = null;
    private String ivDB2InstallDir = null;
    private String ivAppName = null;
    private String ivWASServerName = null;
    private String ivWasProfileName = null;
    private String ivWasVersion = null;
    private String ivScriptsDir = null;
    private SupportWindowsHelper ivHelper = null;

    /**
     * Constructor Retrieve input parameter parm#0 = The response file name.
     */
    public TradeWinPDC(String[] args) {
        // The properties file is in the same format as a response file so
        // manipulate it in the same way as a response file
    }
}

```

```

        setResponseFileName((1 == args.length) ? args[0] : null);
        setMainResources(TradeWinCommon.SAMPLE_MESSAGES);
    }

/**
 * The Trade application requires WAS and DB2. This PDC will check to see if:
 * WAS is installed. (version specified in the properties file)
 * DB2 is installed
 * If both are installed, it will determine if
 * the application is already installed. It will use a jacl script via
 * wsadmin, the results being put into a string - ivCommandResult which will
 * be searched for errors. - if there is a WAS error returned, it could be
 * that WAS is not running, therefore the WAS start command will be run and
 * the check to determine if the app is already installed will be run again. -
 * There are four possible results in the command result string: -
 * APP_EXISTS indicating the application is already installed - return
 * PDC_EXISTS - APP_DOES_NOT_EXIST - return PDC_DOES_NOT_EXIST - WASxxxxx
 * messages indicating the WAS server is not started - return FAILURE - the
 * result file does not exist or some other, unexpected error - return
 * FAILURE
 */
private int check() {
    int rc = SUCCESS;
    int resultrc = FAILURE; // init to return a failure result
    boolean startedWAS = false;

    ivHelper = (SupportWindowsHelper) getHelper();
    setJarFile(ivHelper.getProductInstallingId(this));

    // get the values set in the properties file, properties file is
    // mandatory
    if (getResponseFileName() == null || getResponseFileName().trim().equals("")) {
        setMessage(getResourceString(TradeNLSKeys.NO_PROPERTIES_FILE));
        ivHelper.log(this);
        rc = FAILURE;
    } else {
        rc = getProperties();
    }

    // Determine if WAS is installed and get the WAS directories
    if (rc == SUCCESS) {
        rc = determineWasDir();
    }

    // Determine if DB2 is installed
    if (rc == SUCCESS) {
        String db2InstallDir = null;
        db2InstallDir = TradeWinCommon.determineDB2Dir(this, ivHelper);
        if (db2InstallDir == null || db2InstallDir.trim().equals("")) {
            setMessage(getResourceString(TradeNLSKeys.DB2_NOT_INSTALLED));
            ivHelper.log(this);
            rc = FAILURE;
        }
    }

    if (rc == SUCCESS) {

        // Determine if the application is already installed
        // Run the WebSphere Admin application (wsadmin), the
        // -f option indicates a jacl script is the next arg

```

```

        // After the script name, the arg for the jacl script is:
        // - the application name
        String checkCommand = "cmd /C " + getWasProfileBinDir() + "wsadmin -f " +
ivHelper.getUnpackedDir(this).replace('\\', '/') + getScriptsDir() + TradeWinCommon.CHECK_INSTALL_SCRIPT +
// Script
        // to
        // run
        " " + getAppName(); // the application name

// invoke and log the command
rc = invokeCommand(checkCommand, checkCommand);

if (rc == SUCCESS) {
    // set up directory from which to issues WAS command
    // Search the command result for unique strings indicating a
    // wsadmin error
    // If any of these errors occur, it may be that WAS is not
    // started
    // Start the server using the command in the bin directory of
    // the specified profile.
    if (TradeWinCommon.messageExists(getCommandResult(), TradeWinCommon.WAS_WSADMIN_ERRORS)) {
        String startCommand = "cmd /C " + getWasProfileBinDir() + "startServer " +
getServerName();
        rc = invokeCommand(startCommand, startCommand);
        // Check the result of the start command
        if (TradeWinCommon.messageExists(getCommandResult(), TradeWinCommon.WAS_START_ERRORS))
{
            // error starting the server
            rc = FAILURE;
        } else {
            startedWAS = true;
            // and run the check install script again.
            rc = invokeCommand(checkCommand, checkCommand);
            // check for wsadmin errors
            if (TradeWinCommon.messageExists(getCommandResult(),
TradeWinCommon.WAS_WSADMIN_ERRORS)) {
                rc = FAILURE; // set a FAILURE return code if errors
            }
        }
    }

    // Get the result from the jacl script
    if (rc == SUCCESS) {
        if (getCommandResult().indexOf(TradeWinCommon.APP_EXISTS) >= 0) {
            resultrc = PDC_EXISTS;
            if (startedWAS == true) {
                // put WAS back to its original state
                String stopCommand = "cmd /C " + getWasProfileBinDir() + "stopServer " +
getServerName();
                invokeCommand(stopCommand, stopCommand);
            }
        }

        if (getCommandResult().indexOf(TradeWinCommon.APP_DOES_NOT_EXIST) >= 0) {
            resultrc = PDC_DOES_NOT_EXIST;
        }
    }
}
}
}
}

```

```

        return resultrc;
    }

    /**
     * Get values from the properties file
     */
    private int getProperties() {
        int rc = SUCCESS;
        // The properties file name was set in the constructor

        // Log the contents of the properties file
        ivHelper.logNewLine(this);
        setFileName(getResponseFileName());
        ivHelper.logAppendFile(this);
        ivHelper.logNewLine(this);

        // get the values specified in the properties file that are needed by
        // this PDC

        setKey(TradeWinCommon.WAS_VERSION_KEY); // WAS version is used to get
        // the install path and bin path
        setWasVersion(ivHelper.getResponseFileValue(this));
        if (getWasVersion() == null || getWasVersion().trim().equals("")) {
            setMessage(getResourceString(TradeNLSKeys.NO_PROPERTY, TradeWinCommon.WAS_VERSION_KEY));
            ivHelper.log(this);
            rc = FAILURE;
        }

        setKey(TradeWinCommon.SERVER_NAME_KEY); // server name to use for the
        // start command
        setServerName(ivHelper.getResponseFileValue(this));
        if (getServerName() == null || getServerName().trim().equals("")) {
            setMessage(getResourceString(TradeNLSKeys.NO_PROPERTY, TradeWinCommon.SERVER_NAME_KEY));
            ivHelper.log(this);
            rc = FAILURE;
        }

        setKey(TradeWinCommon.WAS_PROFILE_KEY); // WAS profile name
        setWasProfileName(ivHelper.getResponseFileValue(this));
        if (getWasProfileName() == null || getWasProfileName().trim().equals("")) {
            setMessage(getResourceString(TradeNLSKeys.NO_PROPERTY, TradeWinCommon.WAS_PROFILE_KEY));
            ivHelper.log(this);
            rc = FAILURE;
        }

        setKey(TradeWinCommon.APP_NAME_KEY); // the name of the application to
        // be installed, it is required
        setAppName(ivHelper.getResponseFileValue(this));
        if (getAppName() == null || getAppName().trim().equals("")) {
            setMessage(getResourceString(TradeNLSKeys.NO_PROPERTY, TradeWinCommon.APP_NAME_KEY));
            ivHelper.log(this);
            rc = FAILURE;
        }

        setKey(TradeWinCommon.SCRIPTS_DIR_KEY); // the name of the scripts
        // directory
        setScriptsDir(ivHelper.getResponseFileValue(this));
        if (getScriptsDir() == null || getScriptsDir().trim().equals("")) {
            setScriptsDir(" ");
        } else
    }

```



```

        // Add a slash to the end of the path if one doesn't exist
        if (!getScriptsDir().endsWith("/")) {
            setScriptsDir(getScriptsDir() + "/");
        }

        return rc;
    }

    /**
     * Invoke the command, commandLog is the text to be logged
     *
     * This will set the command result in an instance variable in case the
     * calling routine needs to analyze the result
     *
     * The command and the result will both be logged here
     */
    private int invokeCommand(String command, String commandLog) {
        int rc = SUCCESS;

        try {
            setCommandResult(TradeWinCommon.invokeCommand(this, ivHelper, command, commandLog));
        } catch (Exception e) {
            rc = FAILURE;
            setMessage(getResourceString(TradeNLSKeys.CMD_EXCEPTION, e.toString()));
            ivHelper.log(this);
        }

        return rc;
    }

    /**
     * Get the install and bin directory for WAS from the Windows registry
     * Set the installableApps directory based on the WASinstall directory
     */
    private int determineWasDir() {
        int rc = SUCCESS;

        setWasInstallDir(TradeWinCommon.determineWasDir(this, ivHelper, getWasVersion()));
        if (getWasInstallDir() == null) {
            setMessage(getResourceString(TradeNLSKeys.WAS_NOT_INSTALLED));
            ivHelper.log(this);
            rc = FAILURE;
        } else {
            // Set the WAS profile bin dir
            setWasProfileBinDir(getWasInstallDir() + "profiles/" + getWasProfileName() + "/bin/");
        }

        return rc;
    }

    /**
     *
     * Getters and Setters
     */

    /**
     * Get the command result.
     */
    private String getCommandResult() {
        return ivCommandResult;
    }

```

```

/**
 * Set the command result.
 */
private void setCommandResult(String commandResult) {
    ivCommandResult = commandResult;
}

/**
 * Get the application name.
 */
private String getAppName() {
    return ivAppName;
}

/**
 * Set the application name.
 */
private void setAppName(String appName) {
    ivAppName = appName;
}

/**
 * Get the server name.
 */
private String getServerName() {
    return ivWASServerName;
}

/**
 * Set the server name.
 */
private void setServerName(String serverName) {
    ivWASServerName = serverName;
}

/**
 * Get the WAS profile name.
 */
private String getWasProfileName() {
    return ivWasProfileName;
}

/**
 * Set the WAS profile name.
 */
private void setWasProfileName(String profName) {
    ivWasProfileName = profName;
}

/**
 * Get the WAS version.
 */
private String getWasVersion() {
    return ivWasVersion;
}

/**
 * Set the WAS version
 */

```

```

private void setWasVersion(String ver) {
    ivWasVersion = ver;
}

/**
 * Get the WAS profile bin directory name.
 */
private String getWasProfileBinDir() {
    return ivWASProfileBinDir;
}

/**
 * Set the WAS profile bin directory name.
 */
private void setWasProfileBinDir(String binDir) {
    ivWASProfileBinDir = binDir;
}

/**
 * Get the WAS install directory name.
 */
private String getWasInstallDir() {
    return ivWASInstallDir;
}

/**
 * Set the WAS install directory name.
 */
private void setWasInstallDir(String installDir) {
    ivWASInstallDir = installDir;
}

/**
 * Get the directory within userPrograms where the scripts are located
 */
private String getScriptsDir() {
    return ivScriptsDir;
}

/**
 * Set the directory where the application documents are to be installed.
 */
private void setScriptsDir(String scriptsDir) {
    ivScriptsDir = scriptsDir;
}

/**
 * ***** Main Routine *****
 *
 * @param args
 *      1 - properties file
 */
public static void main(String args[]) {
    TradeWinPDC checker = new TradeWinPDC(args);
    System.exit(checker.check());
}
}

```

The TradeWinCommon.java program

Example A-5 shows the source code of the TradeWinCommon class.

Example: A-5 Source of TradeWinCommon

```
package com.trade;
/**
 * @(#) TradeWinCommon.java
 *
 * Licensed Materials - Property of IBM
 *
 * 5724-J10
 *
 * (C) Copyright IBM Corp. 2005
 *
 * US Government Users Restricted Rights - Use, duplication or disclosure
 * restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * This is sample code made available for use in accordance with terms set
 * forth in the license agreement document for the IBM Express Runtime.
 */

import com.ibm.jsdt.support.SupportWindowsBase;
import com.ibm.jsdt.support.SupportWindowsHelper;

/**
 * This class contains constants and methods referenced by all user programs in the Sample Wrapper.
 */
public class TradeWinCommon {
    public static final String copyright0="Licensed Materials - Property of IBM";
    public static final String copyright1="5724-J10";
    public static final String copyright2="(C) Copyright IBM Corp. 2005 All Rights Reserved.";
    public static final String copyright3="US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with IBM Corp.";

    public static final String WASEXPRESS_NAME = "WebSphere Application Server, Express";
    public static final String SAMPLE_MESSAGES = "com.ibm.iru.tradewin.TradeMessagesNLS";
    public static final String SAMPLE_FILES = "RuntimeDocs";
    public static final String CHECK_INSTALL_SCRIPT = "CheckAppInstall.jacl";
    public static final String APP_EXISTS = "APP_EXISTS"; // used in the checkInstall script
    public static final String APP_DOES_NOT_EXIST = "APP_DOES_NOT_EXIST"; // used in the checkInstall
script
    public static final String[] WAS_WSADMIN_ERRORS = {
        "WASX7023E", // Error creating connection to host
        "WASX7213I", // The scripting client is not connected to a server process
        "WASX7017E"}; // Exception received while running script
    public static final String[] WAS_START_ERRORS = {
        "ADMU0111E"}; // Program exiting with error

    // application response file keys
    public static final String WAS_VERSION_KEY = "WAS.version";
    public static final String SERVER_NAME_KEY = "WAS.serverName";
    public static final String WAS_SCRIPT_KEY = "TRADE.WASscript";
    public static final String APP_NAME_KEY = "WAS.appName";
    public static final String WAS_PROFILE_KEY = "WAS.profile";
    public static final String DB2_SCRIPT_KEY = "TRADE.DB2script";
    public static final String SCRIPTS_DIR_KEY = "TRADE.scriptsDir";
    public static final String DB_NAME_KEY = "DB2.databaseName";
    public static final String DB2_ADMIN_ID_KEY = "DB2.AdminID";
```

```

public static final String DB2_ADMIN_PW_KEY = "DB2.AdminPW";
public static final String JDBC_CLASSPATH_KEY = "WAS.JDBCclasspath";
public static final String WAS_INSTALLABLE_APPS_DIR_KEY = "WAS.installableAppsDir";
public static final String APP_FILE_KEY = "WAS.appInstallFile";
public static final String IHS_SERVER_NAME_KEY = "IHS.webServerService";

/**
 * Invoke the command, commandLog is the command text to be logged
 * Both the command and the command result will be logged
 *
 * The command and the result will both be logged here
 * @param base, SupportWindowsBase
 * @param helper, SupportWindowsHelper
 * @param command
 * @param command text to be logged
 * @return the command result
 */
public static String invokeCommand(SupportWindowsBase base, SupportWindowsHelper helper, String
command, String commandLog) throws Exception
{
    String commandResult = null;
    String text = null;

    // if no command log is specified, log the command itself
    if (commandLog == null || commandLog.trim().equals(""))
        text = command;
    else
        text = commandLog;

    // log the command
    base.setMessage(base.getResourceString(TradeNLSKeys.CMDINVOKED, text));
    helper.log(base);
    helper.logNewLine(base);

    base.setCommand(command);
    // invoke the command and set the command result to be returned
    // this may throw an exception
    commandResult = helper.getSystemCommandOutput(base);
    base.setMessage(commandResult);
    helper.log(base);

    return commandResult;
}

/**
 * Get the install directory for WAS from the Windows registry
 * The returned directory will be in short path format with forward slashes, ending with a slash
 * @param base, SupportWindowsBase
 * @param helper, SupportWindowsHelper
 * @param wasVersion
 * @return the WAS install directory
 */
public static String determineWasDir(SupportWindowsBase base, SupportWindowsHelper helper, String
wasVersion)
{

```

```

String wasInstallDir = null;
// set the WAS registry key
base.setRegistrySubKey("SOFTWARE\\IBM\\" + WASEXPRESS_NAME + "\\" + wasVersion);

if (helper.doesRegKeyExist(base)) {
    // get the WAS install location from the registry
    base.setRegistryStringValue("InstallLocation");
    base.setPath(helper.getRegistryValue(base));
    wasInstallDir = helper.getWindowsShortPath(base);
    // Ensure the path contains forward slashes
    wasInstallDir = wasInstallDir.replace('\\', '/');
    // Add a slash to the end of the path if one doesn't exist
    if (!wasInstallDir.endsWith("/")) {
        wasInstallDir = wasInstallDir + "/";
    }
}

return wasInstallDir;
}

/**
 * Get the install directory for DB2 from the Windows registry
 * The returned directory will be in short path format with forward slashes, ending with a slash
 * @param base, SupportWindowsBase
 * @param helper, SupportWindowsHelper
 * @return the DB2 install directory
 */
public static String determineDB2Dir(SupportWindowsBase base, SupportWindowsHelper helper)
{
    String db2InstallDir = null;
    // Set the DB2 registry key
    String DB2_REGISTRY_KEY = "SOFTWARE\\IBM\\DB2";
    base.setRegistrySubKey(DB2_REGISTRY_KEY);
    base.setRegistryStringValue("DB2 Path Name");

    // get the DB2 install directory from the Windows registry
    if (helper.doesRegKeyExist(base)) {
        db2InstallDir = helper.getRegistryValue(base);
        if (db2InstallDir != null && !db2InstallDir.trim().equals("")) {
            // convert to windows short path format
            base.setPath(db2InstallDir);
            db2InstallDir = helper.getWindowsShortPath(base);
            // Ensure the path contains forward slashes then store the value
            db2InstallDir = db2InstallDir.trim().replace('\\', '/');
        }
    }

    return db2InstallDir;
}

/**
 * Determine if one or more messages exist in a string
 * @param source, the string to check
 * @param msgs, an array of message numbers
 * @return boolean, true if a message is found in the string
 */
public static boolean messageExists(String source, String [] msgs) {
    for (int i = 0; i < msgs.length; i++) {
        if (source.indexOf(msgs[i]) >= 0) {
            return true;
        }
    }
}

```

```

    }
    return false;
}
}

```

The TradeNLSKeys.java program

Example A-6 shows the source code of the TradeNLSKeys class.

Example: A-6 Source of TradeNLSKeys

```

package com.trade;
/**
 * Licensed Materials - Property of IBM
 *
 * 5724-F71 5724-J10
 *
 * (C) Copyright IBM Corp. 2004, 2005
 *
 * US Government Users Restricted Rights - Use, duplication or disclosure
 * restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * This is sample code made available for use in accordance with terms set
 * forth in the license agreement document for the IBM Express Runtime.
 */

/**
 * NLSKeys contains the statics references to the resources
 * <p>
 * Static keys for the Resource Bundles used by the IBM Express Runtime Sample Application
 */

public class TradeNLSKeys {
    private static final String copyright0="Licensed Materials - Property of IBM";
    private static final String copyright1="5724-F71 5724-J10";
    private static final String copyright2="(C) Copyright IBM Corp. 2004, 2005 All Rights Reserved.";
    private static final String copyright3="US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with IBM Corp.";

    public static final String CMDINVOKED = "10000";
    public static final String CMD_EXCEPTION = "10001";
    public static final String COPYFILE_SUCCESS = "10003";
    public static final String COPYFILE_FAIL = "10004";
    public static final String CMD_FAIL = "10010";
    public static final String FILE_CREATE = "10012";
    public static final String FILE_CREATE_EXCEPTION = "10013";
    public static final String FILE_CREATE_FAILED = "10014";
    public static final String NO_PROPERTIES_FILE = "10015";
    public static final String NO_PROPERTY = "10016";
    // -----
    // MESSAGES for WAS - range 1000-1099
    // -----
    public static final String START_WASEXPRESS_FAIL = "11016";
    public static final String STOP_WASEXPRESS_FAIL = "11017";
    public static final String WAS_NOT_INSTALLED = "11018";
    // -----
    // MESSAGES for DB2 - range 1100-1199

```

```

// -----
public static final String DB2_NOT_INSTALLED =          "11100";
// -----
// MESSAGES for IHS - range 1200-1299
// -----
public static final String START_HTTPSERVER_FAIL =      "11200";
public static final String STOP_HTTPSERVER_FAIL =      "11201";
public static final String NO_HTTPSERVER =              "11202";
// -----
// MESSAGES for Sample Application - range 1400-1499
// -----
public static final String INSTALLING_SAMPLE =          "11400";
public static final String NO_WAS_SCRIPT =              "11401";
public static final String NO_DB2_SCRIPT =              "11402";

}

```

The TradeMessagesNLS.java program

Example A-7 shows the source code of the TradeMessagesNLS class.

Example: A-7 Source of TradeMessagesNLS

```

package com.trade;

/**
 * Licensed Materials - Property of IBM
 *
 * 5724-F71 5724-J10
 *
 * (C) Copyright IBM Corp. 2004, 2005
 *
 * US Government Users Restricted Rights - Use, duplication or disclosure
 * restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * This is sample code made available for use in accordance with terms set
 * forth in the license agreement document for the IBM Express Runtime.
 */
import java.util.*;

/**
 * MessagesNLS contains the default US/English literal objects for the IBM Express Runtime Sample
 * Application
 */
public class TradeMessagesNLS extends ListResourceBundle {
    private static final String copyright0="Licensed Materials - Property of IBM";
    private static final String copyright1="5724-F71 5724-J10";
    private static final String copyright2="(C) Copyright IBM Corp. 2004, 2005 All Rights Reserved.";
    private static final String copyright3="US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with IBM Corp.";
    /**
     * Localize the second argument in each pair of brackets.
     * @since JDK1.3
     */
    static final Object[][] messages =
    {
        // -----
        // Common Messages 0-799
        // -----
    }

```



```

// {0} represents a command that was issued to the underlying operating system
{TradeNLSKeys.CMDINVOKED, "IRUS0000: The following command was issued:\n {0}"},
// {0} represents the exception which occurred
{TradeNLSKeys.CMD_EXCEPTION, "IRUS0001: Exception occurred issuing command.\n
Exception: {0}"},
// {0} represents the source file and {1} represents the target file
{TradeNLSKeys.COPYFILE_SUCCESS, "IRUS0003: Copy file {0} to {1} was successful"},
{TradeNLSKeys.COPYFILE_FAIL, "IRUS0004: Copy file {0} to {1} failed"},
{TradeNLSKeys.CMD_FAIL, "IRUS0010: Command failed with return code {0}."},
{TradeNLSKeys.FILE_CREATE, "IRUS0012: Creating file {0}."},
{TradeNLSKeys.FILE_CREATE_EXCEPTION, "IRUS0013: Exception occurred creating file.\n{0}"},
{TradeNLSKeys.FILE_CREATE_FAILED, "IRUS0014: Failed creating file {0}."},
{TradeNLSKeys.NO_PROPERTIES_FILE, "IRUS0015: Properties File not specified"},
{TradeNLSKeys.NO_PROPERTY, "IRUS0016: Required property {0} not specified."},

// -----
// MESSAGES for WAS - range 1000-1099
// -----
{TradeNLSKeys.START_WASEXPRESS_FAIL, "IRUS1016: Failed to start WebSphere Express
Application Server."},
{TradeNLSKeys.STOP_WASEXPRESS_FAIL, "IRUS1017: Failed to stop WebSphere Express
Application Server."},
{TradeNLSKeys.WAS_NOT_INSTALLED, "IRUS1018: WebSphere Express is not installed."},

// -----
// MESSAGES for DB2 - range 1100-1199
// -----
{TradeNLSKeys.DB2_NOT_INSTALLED, "IRUS1100: DB2 is not installed."},

// -----
// MESSAGES for IHS - range 1200-1299
// -----
{TradeNLSKeys.START_HTTPSERVER_FAIL, "IRUS1200: Failed to start IBM HTTP Server."},
{TradeNLSKeys.STOP_HTTPSERVER_FAIL, "IRUS1201: Failed to stop IBM HTTP Server."},
{TradeNLSKeys.NO_HTTPSERVER, "IRUS1202: IBM HTTP Server not specified."},

// -----
// MESSAGES for Sample Application - range 1400-1499
// -----
{TradeNLSKeys.INSTALLING_SAMPLE, "IRUS1400: Installation of the Express Runtime Sample
Application is in progress."},
{TradeNLSKeys.NO_WAS_SCRIPT, "IRUS1401: No WebSphere script was run"},
{TradeNLSKeys.NO_DB2_SCRIPT, "IRUS1402: No DB2 script was run"},
// End Messages translations.
};

/**
 */
public Object[] [] getContents()
{
    return getMessages();
}

/**

```

```

    * Convenience static method to get the messages array. It is
    * public because JUnitMessageAbstraction needs access to this
    * method.
    */
    public static Object[][] getMessages()
    {
        return messages;
    }
}

```

The CheckAppInstall.jacl script file

Example A-8 shows the source of the CheckAppInstall script.

Example: A-8 Source of CheckAppInstall

```

# %I% %W% %G% %U%
# *****

# CheckAppInstall will list all installed apps and put out
# the text APP_EXISTS if the application name in the arg is found or
# the text APP_DOES_NOT_EXIST if the application name is not found
# One arg is expected, the name of the application

global AdminApp
global env
set appList [$AdminApp list]

set appArg [lindex $argv 0]
set result APP_DOES_NOT_EXIST

foreach appName $appList {
    puts $appName
    if {$appName == $appArg} {
        set result APP_EXISTS
        break
    }
}
puts $result

```

The WebSphereConfigProcs.jacl script file

Example A-9 shows the source of the WebSphereConfigProcs script.

Example: A-9 Source of WebSphereConfigProcs

```
# %I% %W% %G% %U%
# Licensed Materials - Property of IBM
#
# 5724-F71 5724-J10
#
# (C) Copyright IBM Corporation 2004, 2005 All Rights Reserved
#
# US Government Users Restricted Rights- Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM
# Corp.
#
# This is sample code made available for use in accordance with
# terms set forth in the license agreement document for the IBM
# Express Runtime.

#-----
# Config procs
#-----

# JACL NOTES
# Braces inside comments are still matched.
# What that means is that you must provide a matching end brace if you want to
# comment out a line e.g.
# if { condition } { ;# you must provide the ending brace even tho this line is commented
# }

#-----
#
# createdB2JDBCProvider - this takes base node, the server name,
#                       the JDBCProvider name, the classpath to the
#                       JDBCProvider code, the name of the implementation
#                       class, the xa setting, and a description
# Parameters:
#   bn - baseNode - typically DefaultNode
#   serv - server name - typically server1
#   provName - name of the JDBC provider - can be any string
#   classpath - path to and including the db2java.zip file e.g. C:/sqllib/java/db2java.zip
#   implClass - implementation class name of the JDBC driver - typically
COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource
#   xa - whether this datasource is single or 2 phase - true/false
#   desc - free form description string
#
# NOTES:
# - nativepath - not used in this proc
# - propertySet - not used in this proc
#
#-----
proc createdB2JDBCProvider {bn serv provName classPath implClass xa desc} {
    puts "\nConfigProcs: createdB2JDBCProvider $bn $serv provName classPath implClass xa desc"
    global AdminConfig ;# Access the AdminConfig command

    if {[file exists $classPath]} {
```

```

        # get the path name upto the driver to use for DB2_JDBC_DRIVER_PATH
        set db2jdbcdriverpath [file dirname $classpath]
        # set the WebSphere environment variable DB2_JDBC_DRIVER_PATH at node scope
        updateVariableMap DB2_JDBC_DRIVER_PATH $db2jdbcdriverpath
        # use the WebSphere environment variable DB2_JDBC_DRIVER_PATH in the JDBC provider
        set jdbcProvClasspath [file join \${DB2_JDBC_DRIVER_PATH}\ [file tail $classpath]]
    } else {
        return -code error "Could not find the JDBC driver at the location provided $classpath"
    }
}

#-----
# Get the config id of the server
#-----
set parent [$AdminConfig getid /Node:$bn/]

#-----
# Set the variables for the JDBCProvider, use path and name provided
# in the arguments
#-----
set pname_attr [list name $provName]
set path_attr [list classpath $jdbcProvClasspath]
set impl_attr [list implementationClassName $implClass]
set desc_attr [list description $desc]
set xa_attr [list xa $xa]
set jdbcAttrs [list $pname_attr $path_attr $impl_attr $xa_attr $desc_attr]
puts "\nConfigProcs: create JDBCProvider $parent $jdbcAttrs"
$AdminConfig create JDBCProvider $parent $jdbcAttrs
}

# createJDBCProviderUsingTemplate - creates a new DB2 type 2 legacy provider at the server scope
#
# This method has the advantage of delegating most of the responsibility for
# knowing the right implementation class name and other driver details to
# WebSphere itself.
#
# Parameters
#     provName - name of the provider - any text string
#     classpath - path to and including the db2java.zip file e.g. C:/sqllib/java/db2java.zip
#     serverName - optional - defaults to server1
#     nodeName - optional - defaults to DefaultNode
#
proc createJDBCProviderUsingTemplate {provName classPath nodeName serverName } {
    puts "\nConfigProcs: createJDBCProviderUsingTemplate $provName $classpath $nodeName $serverName"
    global AdminConfig ;# Access the AdminConfig command

    if {[file exists $classpath]} {
        # get the path name upto the driver to use for DB2_JDBC_DRIVER_PATH
        set db2jdbcdriverpath [file dirname $classpath]
        # set the WebSphere environment variable DB2_JDBC_DRIVER_PATH at node scope
        # updateVariableMap DB2_JDBC_DRIVER_PATH $db2jdbcdriverpath $serverName $nodeName
        updateVariableMap DB2UNIVERSAL_JDBC_DRIVER_PATH $db2jdbcdriverpath $serverName $nodeName
    } else {
        return -code error "Could not find the JDBC driver at the location provided $classpath"
    }

    set parent [$AdminConfig getid /Node:$nodeName/]
    set pname_attr [list name $provName]
    set attrList [list $pname_attr]
    # set templ [$AdminConfig listTemplates JDBCProvider "DB2 Legacy CLI-based Type 2 JDBC Driver("]

```

```

set templ [$AdminConfig listTemplates JDBCProvider "DB2 Universal JDBC Driver Provider ("
#$AdminConfig createUsingTemplate JDBCProvider $parent {{name $provName}} $templ
puts " "
puts "ConfigProcs: AdminConfig createUsingTemplate JDBCProvider $parent $attrList $templ"
puts " "
$AdminConfig createUsingTemplate JDBCProvider $parent $attrList $templ

# Create by template automatically creates a 4.0 and 5.0 DS with no name
# We could modify them to suit us but it adds to the complexity of the main script
# so just delete the extra template datasources
# ignore any errors

catch [set dsToDelete [$AdminConfig getid /JDBCProvider:$provName/DataSource:]]
catch [$AdminConfig remove $dsToDelete]

catch [set dsToDelete [$AdminConfig getid /JDBCProvider:$provName/WAS40DataSource:]]
catch [$AdminConfig remove $dsToDelete]

puts "Please remember to issue a \AdminConfig save if you wish to save this configuration change"
}

#-----
# createJAASDataAuth - this is used for connection to the
#                   database and authorization for the database
#
# JAASDataAuth attributes are:
# "alias String"
# "description String"
# "password String"
# "userId String"
#-----
proc createJAASDataAuth {aliasName user pw desc} {
    global AdminConfig

    set security [$AdminConfig list Security]
    set jaasAlias_attr [list alias $aliasName]
    set jaasDesc_attr [list description $desc]
    set user_attr [list userId $user]
    set pw_attr [list password $pw]
    set jaas_attrs [list $jaasAlias_attr $user_attr $pw_attr $jaasDesc_attr]
    # create special attributes to be logged - we do not want to log the password
    set pw_attr_log [list password ****]
    set jaas_attrs_log [list $jaasAlias_attr $user_attr $pw_attr_log $jaasDesc_attr]
    puts " "
    puts "ConfigProcs: AdminConfig create JAASAuthData $security $jaas_attrs_log"
    puts " "
    $AdminConfig create JAASAuthData $security $jaas_attrs
}

#-----
#
# createdB2DataSource - this takes base node, the server name,
#                   the JDBCProvider name to which this DataSource
#                   is to be associated,
# NOTE: A ConnectionPool will also be created within this proc
#

```

```

# DataSource attributes are:
# "authDataAlias String"
# "authMechanismPreference ENUM(BASIC_PASSWORD, KERBEROS)"
# "category String"
# "connectionPool ConnectionPool"
# "datasourceHelperClassname String"
# "description String"
# "jndiName String"
# "mapping MappingModule"
# "name String"
# "provider J2EEResourceProvider@"
# "relationalResourceAdapter J2CResourceAdapter@"
# "statementCacheSize Integer"
# "propertySet J2EEResourcePropertySet"
#
# NOTES:
#   - mapping is not used
#   - we will set the JNDI name as "jdbc/" with the datasource name appended
#   - the property set for a DB2 DataSource includes
#     database name, this is a required attribute
#   - a relational resource adapter will be created using
#     the "WebSphere Relational Resource Adapter"
#
#-----
proc createDB2DataSource {bn serv JDBCProvName dsName dbName dsHelper authAlias authMech cat desc} {
    global AdminConfig

    set jdbcProvId [AdminConfig getid /JDBCProvider:$JDBCProvName/]

    #-----
    # Set up the properties for a DB2 DataSource
    #-----
    set dbname_attr [list [list name databaseName] [list value $dbName] [list type java.lang.String] [list
required true] [list description "The DB2 database name"]]
    set newprops [list $dbname_attr]
    set resprops [list resourceProperties $newprops]
    set dsProp_attrs [list propertySet [list $resprops]]
    #-----
    # Set up the attributes for a connection pool
    #-----
    # ConnectionPool attributes are:
    # "agedTimeout Long"
    # "connectionTimeout Long"
    # "maxConnections Integer"
    # "minConnections Integer"
    # "purgePolicy ENUM(EntirePool, FailingConnectionOnly)"
    # "reapTime Long"
    # "unusedTimeout Long"
    set agedTimeout_attr [list agedTimeout "0"]
    set connectionTimeout_attr [list connectionTimeout "1800"]
    set maxConnections_attr [list maxConnections "10"]
    set minConnections_attr [list minConnections "1"]
    set purgePolicy_attr [list purgePolicy "EntirePool"]
    set reapTime_attr [list reapTime "180"]
    set unusedTimeout_attr [list unusedTimeout "1800"]
    set connPool_attrs [list connectionPool [list $agedTimeout_attr $connectionTimeout_attr
$maxConnections_attr $minConnections_attr $purgePolicy_attr $reapTime_attr $unusedTimeout_attr]]
    #-----
    # Set up a relational resource adapter.
    #-----

```

```

set rra [$AdminConfig getid "/Node:$bn/J2CResourceAdapter:WebSphere Relational Resource Adapter/"]
set rra_attr [list relationalResourceAdapter $rra]
#-----
# Set up DataSource Attributes
#-----
set name_attr [list name $dsName]
set cat_attr [list category $cat]
set prov_attr [list provider $jdbcProvId]
set jndiName [concat jdbc/$dsName]
set jndiName_attr [list jndiName $jndiName]
set desc_attr [list description $desc]
set authmech_attr [list authMechanismPreference $authMech]
set authDataAlias_attr [list authDataAlias $authAlias]
set authMap_attr [list mapping [list [list authDataAlias $authAlias] [list mappingConfigAlias
DefaultPrincipalMapping]]]
set cache_attr [list statementCacheSize "10"]
set dsHelper_attr [list datasourceHelperClassname $dsHelper]
set ds_attrs [list $name_attr $jndiName_attr $dsHelper_attr $authmech_attr $authDataAlias_attr
$authMap_attr $desc_attr $cat_attr $connPool_attrs $rra_attr $cache_attr $dsProp_attrs]
puts " "
puts "ConfigProcs: AdminConfig create DataSource $jdbcProvId $ds_attrs"
puts " "
$AdminConfig create DataSource $jdbcProvId $ds_attrs
}

#-----
#
# save Configuration
#-----
proc saveConfiguration {} {
    global AdminConfig
    puts "\nConfigProcs: Saving the configuration"
    $AdminConfig save
}

#-----
#
# install Enterprise App - this takes the server and the source ear file for the target application
#                          the appname is only used for naming the application
# return 6 on failure
#-----
proc installEar {server sourceEarFile appname} {
    global AdminApp
    puts "ConfigProcs: installApp for $server $sourceEarFile $appname"

    if {[catch {$AdminApp install "$sourceEarFile" [subst {-appname $appname -server $server}]} result]} {
        puts stderr "\nAn error occurred while installing the Enterprise application contained in"
        puts stderr "$sourceEarFile"
        return -code error -errorcode 6 $result
    } else {
        puts "\nThe Enterprise application was installed successfully"
        puts "In order to use the application please start it first"
    }
}

#-----
#
# install Web App - this takes the server and the source war file for the target application

```

```

#           appname is only used for naming the application
#           contextRoot is used to define the URL at which the application is visible
#           e.g. for http://localhost:7080/myownurl you would use myownurl for the context root
# return 5 on failure
#-----
proc installWar {server sourceWarFile appname contextRoot} {
    global AdminApp
    puts "ConfigProcs:  installWar for $server $sourceWarFile $appname $contextRoot"

    if {[catch {$AdminApp install "$sourceWarFile" [subst {-contextroot $contextRoot -appname $appname
-server $server -usedefaultbindings]]} result]} {
        puts stderr "\nAn error occurred while installing the Web application contained in"
        puts stderr "$sourceWarFile"
        return -code error -errorcode 5 $result
    } else {
        puts "\nThe Web application was installed successfully"
        puts "In order to use the application please start it first"
    }
}

#-----
# installApp - Install the specified application ear file if an
#               application with the same name does not exist.
#-----
proc installApp {appName ear deployejb deployws defaultBindings earMetaData dbType target} {
    #   appName      - application name
    #   ear           - ear file
    #   deployejb    - deploy ejb (true|false)
    #   deployws     - deploy webservices (true|false)
    #   defaultBindings - use default binding (true|false)
    #   earMetaData  - use MetaData from ear (true|false)
    #   dbType       - ejb deploy db type
    #   target[0]    - node name or cluster name
    #   target[1]    - server name

    global AdminControl
    global AdminApp

    puts ""
    puts "Installing application {$appName}..."

    # Check if the application already exists

    set appList [$AdminApp list]
    foreach item $appList {
        if {[string first $appName $item] >= 0} {
            set app $item
            break
        }
    }

    if {[info exists app]} {
        puts "  Application Name:      {$appName}"
        puts "  Ear file:                {$ear}"
        if {[llength $target] == 1} {
            puts "  Target Cluster:          [lindex $target 0]"
        } else {
            puts "  Target Node:             [lindex $target 0]"
            puts "  Target Server:           [lindex $target 1]"
        }
    }
}

```



```

    }
    puts "  Deploy EJB:           ${deployejb}"
    puts "  Deploy WebServices:    ${deployws}"
    puts "  Use default bindings:    ${defaultBindings}"
    puts "  Use Ear MetaData:        ${earMetaData}"
    puts "  Deployed DB Type:        ${dbType}"

    set parms "-appname $appName"
    if {$deployejb == "true"} {
        append parms " -deployejb"
        append parms " -deployejb.dbtype $dbType"
    }
    if {$deployws == "true"} {
        append parms " -deployws"
    }
    if {$defaultBindings == "true"} {
        append parms " -usedefaultbindings"
    }
    if {$earMetaData == "true"} {
        append parms " -useMetaDataFromBinary yes"
    } else {
        append parms " -useMetaDataFromBinary no"
    }
    }

    if {[length $target] == 1} {
        append parms " -cluster [lindex $target 0]"
    } else {
        append parms " -node [lindex $target 0] -server [lindex $target 1]"
    }
    }

    set parms1 [subst {$parms}]

    puts "Starting application install..."

    set app [$AdminApp install $ear $parms1]

    puts "Install completed successfully!"
} else {
    puts "${appName} already exists!"
}
}

return $app
}

#-----
#
# createVirtualHost - this takes base node and a hostname
#
# VirtualHost Attributes are:
# "aliases HostAlias*"
# "mimeTypes MimeEntry*" - we will take the defaults for mimeTypes
# "name String"
#-----
proc createVirtualHost {bn hostName} {
    global AdminConfig
    set vh_parent [$AdminConfig getid /Node:$bn/]
    set vh_name_attr [list name $hostName]
    set vh_attrs [list $vh_name_attr]
    # create the virtual host

```

```

    puts " "
    puts "ConfigProcs: AdminConfig create VirtualHost $vh_parent $vh_attrs"
    puts " "
    $AdminConfig create VirtualHost $vh_parent $vh_attrs
}

```

```

#-----
#
# createHostAlias - this takes the virtual hostname, and an alias
#                   with the associated port numbers as arguments
#                   It is common to have the alias be *
#
# HostAlias Attributes are:
# "hostname String"
# "port String"
#-----

```

```

proc createHostAlias {vHostName alias port} {
    global AdminConfig
    set vhost [$AdminConfig getid /VirtualHost:$vHostName/]
    # set up alias
    set h_attr [list hostname $alias]
    set p_attr [list port $port]
    set alias_attr [list $h_attr $p_attr]
    set vh_cmdAttrs [list [list aliases [list $alias_attr]]]
    # modify the virtual host, creating the new alias
    puts " "
    puts "ConfigProcs: AdminConfig modify $vhost $vh_cmdAttrs"
    puts " "
    $AdminConfig modify $vhost $vh_cmdAttrs
}

```

```

#-----
#
# createHTTPTransport - this takes base node, the server name to contain
#                       these transports, the port number, and
#                       (optionally) the ssl settings to be used
# NOTE: If ssl settings not provided, sslEnabled will be set to false
#
# HTTPTransport Attributes are:
# "address EndPoint"
# "external Boolean"
# "properties Property(TypedProperty)*"
# "sslConfig String"
# "sslEnabled Boolean"
#-----

```

```

proc createHTTPTransport {bn serv host port sslSettings} {
    global AdminConfig

    set parent [$AdminConfig getid /Node:$bn/]

    # Identify the server and assign it to the server variable
    set server [$AdminConfig getid /Node:$bn/Server:$serv/]

    # Identify the Web container belonging to the server and assign it to the wc variable.
    set wc [$AdminConfig list WebContainer $server]

    #Determine if this transport is to be sslEnabled
    if {($sslSettings == "")} {

```

```

set ssl [list sslEnabled false]
} else {
set ssl [list sslConfig $sslSettings sslEnabled true]
}
# set host-port
set h_attr [list host $host]
set p_attr [list port $port]
set endPoint_attr [list $h_attr $p_attr]

#Create HTTP Transport
set transpAddr1 [list [list address $endPoint_attr] $ssl]
puts " "
puts "ConfigProcs: AdminConfig create HTTPTransport $wc $transpAddr1"
puts " "
$AdminConfig create HTTPTransport $wc $transpAddr1
}

#
# dumpStack - a utility method to dump information about the error condition
# Parameters
# result - result of the previous command, just a string.
#
# typical use
# if {[catch {my command} result]} {
#     dumpStack $result
#     <recover from error or exit>
# }
proc dumpStack {result} {
    global errorInfo
    global errorCode
    puts stderr "ConfigProcs: _____Trace_____ "
    puts stderr "[clock format [clock seconds]] "
    puts stderr "Error Code=$errorCode"
    puts stderr $result
    puts stderr $errorInfo
}

# Start an installed application
proc startApp { myApplication server } {
    global AdminControl
    puts "ConfigProcs: startApp $myApplication $server"
    set appManager [$AdminControl queryNames type=ApplicationManager,process=$server,*]
    $AdminControl invoke $appManager startApplication $myApplication
}

# Stop an installed application
proc stopApp { myApplication server } {
    global AdminControl
    puts "ConfigProcs: stopApp $myApplication $server"
    set appManager [$AdminControl queryNames type=ApplicationManager,process=$server,*]
    $AdminControl invoke $appManager stopApplication $myApplication
}

# getMessage - a procedure that extracts the NLS message in the same resource bundle

```

```

#           as that used by the other java parts in the sample.
#
# parameters - key - a key defined in NLSKeys that will be used to look up the message in MessagesNLS /_en
# /_fr etc
#           - args - variable number of arguments that can be provided as replacement parameters
#                   e.g. if key = COPYFILE_FAIL , message = IRU00004: Copy file {0} to {1} failed
#                   and you invoke getMessage COPYFILE_FAIL gaga baba, you would get back
#                   IRU00004: Copy file gaga to baba failed
#
# Several advanced techniques are used here so please be careful
#
# the java classes must be in the classpath. You can see the CP by invoking this in wsadmin
# wsadmin> foreach {key value} [array get env] { puts $key=$value }
# set the classpath like this
# wsadmin -wsadmin_classpath "C:\Program Files\IBM\MidMarketRuntime\SolutionEnabler\unpacked"
#
#   java::new to create a new object as in
#   set x [java::new String "abcd"]
#   java::call to call a static method as
#   set abs_x [java::call Math.abs x]
#   java::field get the value of a class or instance variable as in
#   set max_int [java::field Integer MAX_VALUE]
#   java::null for setting the null value as in
#   set x [java::null]
#   java::isnull tests for null as in
#   if {[java::isnull $x]} ...
#
proc getMessage {key args} { ;# args is a variable argument list
  if {[string first "Linux" $env(os.name)] == 0} {
    set msgkey [java::field com.ibm.iru.samplelnx.NLSKeys $key]
    set defaultBundle [java::call java.util.ResourceBundle getBundle com.ibm.iru.samplelnx.MessagesNLS]
  } else {
    set msgkey [java::field com.ibm.iru.samplewin.NLSKeys $key]
    set defaultBundle [java::call java.util.ResourceBundle getBundle com.ibm.iru.samplewin.MessagesNLS]
  }
  set jnk [$defaultBundle getString $msgkey]
  set attrs [java::new {Object[]} [llength $args] $args ]
  set msg [java::call java.text.MessageFormat format $jnk $attrs]
  return -code ok $msg
}

# Modify Variable Map entries e.g. DB2_JDBC_DRIVER_PATH
# This should be used for classpath for the JDBC provider instead of a direct path to the
# driver. Procedure for later releases.
#
# $AdminConfig showall $varMap
#
proc updateVariableMap {key newValue serverName nodeName} {

  puts "ConfigProc: updateVariableMap $key $newValue $serverName $nodeName"
  global AdminConfig
  # setting WebSphere variables at node level
  set varMap [$AdminConfig getid /Node:$nodeName/VariableMap:]
  set subst [lindex [$AdminConfig showAttribute $varMap entries] 0]

  set foundit "false"

  foreach sub $subst {
    set varName [$AdminConfig showAttribute $sub symbolicName]

```

```

        if {$varName == $key} {
            if {$newValue != ""} {
                puts "Setting $key to $newValue"
                $AdminConfig modify $sub [subst {{value "$newValue"}}]
                set foundit "true"
                break
            }
        }
    }
}

#let us create one if none found
if {$foundit == "false"} {
    # create a list of items
    set nameattr1 [list symbolicName $key]
    set valattr1 [list value $newValue]
    set attr1 [list $nameattr1 $valattr1]
    set attrs [list $attr1]

    puts "creating $key with value of $newValue"
    $AdminConfig modify $varMap [subst {{entries {$attrs}}}]
}
}

# securedB2 - this is a way to allow WebSphere to access DB2 without having to source the db2profile
#             you only need to do this if sourcing the db2profile script in your .profile
#             is not acceptable to you.
# parameters
# db2home - installed location of db2 e.g. /opt/IBM/db2/V8.1
# instance - the db2instance to be used e.g. db2inst1
# server - your server name - optional - defaults to server1 if not provided
#
#
# All you need to do is get the db2 environment variables configured for the JVM .
# It is the only way for the Type 2 driver to get to the native libraries.
# The DB2INSTANCE variable will probably always have to be set, but if you are using ONLY the new
# Type 4 driver then you don't have to set LD_LIBRARY_PATH or LIBPATH as the native libraries aren't
# needed.
#
# The following 3 variables will be added to the java invocation of the app server
#
# LD_LIBRARY_PATH=/home/db2inst1/sqllib/lib:/home/db2inst1/sqllib/bin
# LIBPATH=/home/db2inst1/sqllib/lib:/home/db2inst1/sqllib/bin
# DB2INSTANCE=db2inst1 (or whatever your DB2 instance user name is).

proc securedB2 {bn db2home instance {server server1}} {
    puts "ConfigProcs: securedB2 $bn $db2home $instance $server"
    global AdminConfig
    set appServer [$AdminConfig getid /Node:$bn/Server:$server/]
    #set jvm [$AdminConfig list JavaVirtualMachine $appServer]
    # $AdminConfig modify $jvm $attr
    set attr [subst {{environment {{(name "DB2INSTANCE"){required false}{value "$instance"}} {{(name
"LD_LIBRARY_PATH"){required false}{value "$db2home/lib"}} {{(name "LIBPATH"){required false}{value
"$db2home/lib"}}}}}]]
    set process [$AdminConfig list ProcessDef $appServer]
    $AdminConfig modify $process $attr
    foreach variableSubstitution [$AdminConfig list VariableSubstitutionEntry] {
        foreach varSubstitutionEntry [$AdminConfig showall $variableSubstitution] {
            if { [lsearch -regexp $varSubstitutionEntry "DB2_JDBC_DRIVER_PATH"] >= 0 } {

```

```

        set valueVar [list value "$db2home/java"]
        set varSubAttribs [list $valueVar]
        $AdminConfig modify $variableSubstitution $varSubAttribs
        break
    }
}
}
}

```

```

#-----
# createSIBus - Create a new SIBus if one does not exist. Otherwise,
#               return the existing SIBus.
#-----
proc createSIBus {busName authAlias} {
    #   busName   - SIBus name
    #   authAlias  - authentication alias name

    global AdminTask
    global AdminConfig

    puts " "
    puts "Creating SIBus ${busName}..."

    # Check if the SIBus already exists

    set SIBus [$AdminConfig getid "/SIBus:${busName}/"]

    if {$SIBus == ""} {
        set parms [list -bus $busName -interEngineAuthAlias $authAlias]
        if {[catch {set SIBus [$AdminTask createSIBus $parms]} result]} {
            puts "WSADMIN EXCEPTION: ${result}"
            puts "Terminating due to exception!"
            exit
        } else {
            puts "${busName} created successfully!"
        }
    } else {
        puts "${busName} already exists!"
    }

    return $busName
}

```

```

#-----
# addSIBusMember - Add the specified server or cluster to the
#                  SIBus if it does not already exist. Assumes that the
#                  specified SIBus already exists.
#-----
proc addSIBusMember {busName defaultDS dsJndi optArgs} {
    #   busName   - SIBus name
    #   defaultDS  - create default DS (true|false)
    #   dsJndi     - jndi name of the datasource (only used if defaultDS = false)
    #   optArgs[0] - cluster name or node name
    #   optArgs[1] - server name

    global AdminTask
    global AdminConfig

```

```

puts " "
if {[length $optArgs] == 1} {
    set clusterName [lindex $optArgs 0]
    puts "Adding SIBus member ${clusterName}..."
} else {
    set clusterName "none"
    set nodeName [lindex $optArgs 0]
    set serverName [lindex $optArgs 1]
    puts "Adding SIBus member ${nodeName} - ${serverName}..."
}

puts " Default DataSource:    ${defaultDS}"
if {$defaultDS == "false"} {
    puts " Datasource JNDI Name: ${dsJndi}"
}

# Check if the bus member already exists

set parms [list -bus $busName]
set busMembers [$AdminTask listSIBusMembers $parms]

foreach item $busMembers {
    set cluster [$AdminConfig showAttribute $item cluster]
    set node [$AdminConfig showAttribute $item node]
    set server [$AdminConfig showAttribute $item server]

    if {$cluster == $clusterName || ($server == $serverName && $node == $nodeName)} {
        set member $item
        break
    }
}

if {[length $optArgs] == 1} {
    set parms [list -bus $busName -cluster $clusterName -createDefaultDataSource $defaultDS]
} else {
    set parms [list -bus $busName -node $nodeName -server $serverName -createDefaultDataSource
$defaultDS]
}

if {$defaultDS == "false"} {
    lappend parms -datasourceJndiName $dsJndi
}

if {[info exists member]} {
    if {[catch {set member [$AdminTask addSIBusMember $parms]} result]} {
        puts "WSADMIN EXCEPTION: ${result}"
        puts "Terminating due to exception!"
        exit
    } else {
        puts "SIBus member added successfully!"
    }
} else {
    puts "Bus member already exists!"
}

return $member
}

#-----

```

```

# createSIBDestination - Create a new SIB Destination if one with the same
# name does not exist on the specified SIBus. Otherwise,
# return the existing Destination.
#-----
proc createSIBDestination {SIBus destName destType reliability optArgs} {
    # SIBus      - SIBus name
    # destName   - destination name
    # destType   - destination type
    # reliability - reliability
    # optArgs[0] - cluster name or node name
    # optArgs[1] - server name

    global AdminTask
    global AdminConfig

    if {[llength $optArgs] == 1} {
        set clusterName [lindex $optArgs 0]
    } else {
        set nodeName    [lindex $optArgs 0]
        set serverName   [lindex $optArgs 1]
    }

    puts " "
    puts "Creating SIB Destination ${destName}..."

    # Check if the SIB Destination already exists

    set parms [list -bus $SIBus]
    set destList [$AdminTask listSIBDestinations $parms]

    foreach item $destList {
        set ident [$AdminConfig showAttribute $item identifier]
        if {$ident == $destName} {
            set dest $item
            break
        }
    }

    if {[info exists dest]} {
        puts " Destination Name:  ${destName}"
        puts " Destination Type:  ${destType}"
        puts " Reliability:      ${reliability}"
        if {$destType == "Queue"} {
            if {[llength $optArgs] == 1} {
                puts " Cluster Name:      ${clusterName}"
            } else {
                puts " Node Name:          ${nodeName}"
                puts " Server Name:       ${serverName}"
            }
        }
    }

    set parms [list -bus $SIBus -name $destName -type $destType -reliability $reliability]
    if {$destType == "Queue" && [llength $optArgs] == 1} {
        lappend parms -cluster $clusterName
    } elseif {$destType == "Queue"} {
        lappend parms -node $nodeName -server $serverName
    }

    if {[catch {set dest [$AdminTask createSIBDestination $parms]} result]} {
        puts "WSADMIN EXCEPTION: ${result}"
    }
}

```



```

        puts "Terminating due to exception!"
        exit
    } else {
        puts "${destName} created successfully!"
    }
} else {
    puts "$destName already exists!"
}

return $dest
}

```

```

#-----
# createJMSConnectionFactory - Create a new JMS Connection Factory
#           if one with the same name does not exist on the SIBus.
#           Otherwise, return the existing Connection Factory.
#-----
proc createJMSConnectionFactory {SIBus cfName cfType jndiName authAlias scope} {
    # Create JMS Connection Factory
    #   SIBus      - SIBus name
    #   cfName     - connection factory name
    #   cfType     - connection factory type
    #   jndiName   - connection factory jndi name
    #   authAlias  - authentication alias name
    #   scope      - scope

    global AdminTask

    puts " "
    puts "Creating JMS ${cfType} Connection Factory ${cfName}..."

    # Check if the connection factory already exists

    set parms [list -type $cfType]
    set cfList [$AdminTask listSIBJMSConnectionFactories $scope $parms]

    foreach item $cfList {
        if {[string first $cfName $item] >= 0} {
            set connectionFactory $item
            break
        }
    }

    if {![info exists connectionFactory]} {
        puts "  Connection Factory Name:  ${cfName}"
        puts "  Connection Factory Type:  ${cfType}"
        puts "  JNDI Name:                  ${jndiName}"

        set params [list -name $cfName -jndiName $jndiName -busName $SIBus -type $cfType -authDataAlias
$authAlias]
        if {[catch {set connectionFactory [$AdminTask createSIBJMSConnectionFactory $scope $params]} result]}
        {
            puts "WSADMIN EXCEPTION: ${result}"
            puts "Terminating due to exception!"
            exit
        } else {
            puts "${cfName} created successfully!"
        }
    }
}

```

```

    } else {
        puts "$cfName already exists!"
    }

    return $connectionFactory
}

```

```

#-----
# createJMSQueue - Create a new JMS Queue if one with the same
#                 name does not exist at the specified scope. Otherwise,
#                 return the existing JMS Queue.
#-----
proc createJMSQueue {qName jndiName SIBDest delMode scope} {
    #   qName   - queue name
    #   jndiName - queue jndi name
    #   SIBDest  - SIB destination
    #   delMode  - delivery mode
    #   scope    - scope

    global AdminTask

    puts " "
    puts "Creating JMS Queue ${qName}..."

    # Check if the queue already exists

    set qList [$AdminTask listSIBJMSQueues $scope]
    foreach item $qList {
        if {[string first $qName $item] >= 0} {
            set queue $item
            break
        }
    }

    if {[info exists queue]} {
        puts " Queue Name:      ${qName}"
        puts " JNDI Name:      ${jndiName}"
        puts " SIB Destination:  ${SIBDest}"
        puts " Delivery Mode:    ${delMode}"

        set params [list -name $qName -jndiName $jndiName -queueName $SIBDest -deliveryMode $delMode]
        if {[catch {set queue [$AdminTask createSIBJMSQueue $scope $params]} result]} {
            puts "WSADMIN EXCEPTION: ${result}"
            puts "Terminating due to exception!"
            exit
        } else {
            puts "${qName} created successfully!"
        }
    } else {
        puts "$qName already exists!"
    }

    return $queue
}

```

```

#-----
# createJMSTopic - Create a new JMS Topic if one with the same
#                 name does not exist at the specified scope. Otherwise,

```

```

#          return the existing JMS Topic.
#-----
proc createJMSTopic {tName jndiName tSpace delMode scope} {
    #    tName      - topic name
    #    jndiName   - topic jndi name
    #    tSpace     - topic space
    #    delMode    - delivery mode
    #    scope      - scope

    global AdminTask

    puts " "
    puts "Creating JMS Topic ${tName}..."

    # Check if the topic already exists

    set tList [$AdminTask listSIBJMSTopics $scope]
    foreach item $tList {
        if {[string first $tName $item] >= 0} {
            set topic $item
            break
        }
    }

    if {[info exists topic]} {
        puts "  Topic Name:      ${tName}"
        puts "  JNDI Name:         ${jndiName}"
        puts "  Topic Space:       ${tSpace}"
        puts "  Delivery Mode:     ${delMode}"

        set params [list -name $tName -jndiName $jndiName -topicName $tName -topicSpace $tSpace -deliveryMode
$delMode]
        if {[catch {set topic [$AdminTask createSIBJMSTopic $scope $params]} result]} {
            puts "WSADMIN EXCEPTION: ${result}"
            puts "Terminating due to exception!"
            exit
        } else {
            puts "${tName} created successfully!"
        }
    } else {
        puts "${tName} already exists!"
    }

    return $topic
}

#-----
# createMDBActivationSpec - Create a new MDB Activation Spec if one
#                          with the same name does not exist at the specified
#                          scope. Otherwise, return the existing Activation Spec.
#-----
proc createMDBActivationSpec {mdbName jndiName SIBus JMSDestJndi destType authAlias scope} {
    #    mdbName     - MDB name
    #    jndiName    - activation spec jndi name
    #    SIBus       - SIBus name
    #    JMSDestJndi - JMS destination JNDI name
    #    destType    - destination type
    #    authAlias   - authentication alias name
    #    scope       - scope

```

```

global AdminTask

puts " "
puts "Creating MDB Activation Spec ${mdbName}..."

# Check if the activation spec already exists

set asList [$AdminTask listSIBJMSActivationSpecs $scope]
foreach item $asList {
    if {[string first $mdbName $item] >= 0} {
        set mdb $item
        break
    }
}

if {[info exists mdb]} {
    puts "  MDB Activation Spec Name:    ${mdbName}"
    puts "  JNDI Name:                      ${jndiName}"
    puts "  JMS Destination JNDI Name:    ${JMSDestJndi}"
    puts "  Destination Type:              ${destType}"

    set params [list -name $mdbName -jndiName $jndiName -busName $SIBus -destinationJndiName $JMSDestJndi
-destinationType $destType -authenticationAlias $authAlias]
    if {[catch {set mdb [$AdminTask createSIBJMSActivationSpec $scope $params]} result]} {
        puts "WSADMIN EXCEPTION: ${result}"
        puts "Terminating due to exception!"
        exit
    } else {
        puts "${mdbName} created successfully!"
    }
} else {
    puts "$mdbName already exists!"
}

return $mdb
}

#-----
# createJDBCProvider - Create a new JDBC Provider if one with the
#                      same name does not exist in the specified scope. Otherwise,
#                      return the existing JDBCProvider. The 3 types or providers
#                      currently supported include DB2 JCC, DB2 CLI, and Oracle.
#-----
proc createJDBCProvider {providerType path XA scope} {
    # providerType - provider type (db2|db2cli|oracle|cloudscape)
    # path         - driver classpath path
    # XA           - XA (true|false)
    # scope        - scope

global AdminConfig

# Determine properties based on providerType

if {${providerType} == "db2"} {
    if {${XA} == "true"} {
        set providerName      "DB2 Universal JDBC Driver Provider (XA)"
        set implementationClassName "com.ibm.db2.jcc.DB2XADataSource"
    } else {

```

```

        set providerName          "DB2 Universal JDBC Driver Provider"
        set implementationClassName "com.ibm.db2.jcc.DB2ConnectionPoolDataSource"
    }
} elseif (${providerType} == "oracle") {
    if (${XA} == "true") {
        set providerName          "Oracle JDBC Driver (XA)"
        set implementationClassName "oracle.jdbc.xa.client.OracleXADataSource"
    } else {
        set providerName          "Oracle JDBC Driver"
        set implementationClassName "oracle.jdbc.pool.OracleConnectionPoolDataSource"
    }
} elseif (${providerType} == "db2cli") {
    if (${XA} == "true") {
        set providerName          "DB2 Legacy CLI-based Type 2 JDBC Driver (XA)"
        set implementationClassName "COM.ibm.db2.jdbc.DB2XADataSource"
    } else {
        set providerName          "DB2 Legacy CLI-based Type 2 JDBC Driver"
        set implementationClassName "COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource"
    }
} elseif (${providerType} == "cloudscape") {
    if (${XA} == "true") {
        set providerName          "Cloudscape JDBC Provider (XA)"
        set implementationClassName "com.ibm.db2j.jdbc.DB2jXADataSource"
    } else {
        set providerName          "Cloudscape JDBC Provider"
        set implementationClassName "com.ibm.db2j.jdbc.DB2jConnectionPoolDataSource"
    }
}
}

puts " "
puts "Creating JDBC Provider ${providerName}..."

# Check if the JDBC provider already exists

set name [getName $scope]
set stIndex [expr [string first "|" $scope] + 1]
set endIndex [expr [string first "." $scope] - 1]
set type [string range $scope $stIndex $endIndex]

if (${type} == "cell") {
    set provider [AdminConfig getid "/Cell:$name/JDBCProvider:\"$providerName\"/" ]
} elseif (${type} == "node") {
    set provider [AdminConfig getid "/Node:$name/JDBCProvider:\"$providerName\"/" ]
} elseif (${type} == "server") {
    set provider [AdminConfig getid "/Server:$name/JDBCProvider:\"$providerName\"/" ]
}

if {$provider == ""} {
    puts "  Provider Name:      ${providerName}"
    puts "  Implementation Class: ${implementationClassName}"
    puts "  XA enabled:            ${XA}"

    set attrs [subst {{classpath "$path" {implementationClassName $implementationClassName} {name
"$providerName"} {providerType "$providerName"} {description "$providerName"} {xa "XA"}}}]
    if {[catch {set provider [AdminConfig create JDBCProvider $scope $attrs]} result]} {
        puts "WSADMIN EXCEPTION: ${result}"
        puts "Terminating due to exception!"
        exit
    } else {
        puts "${providerName} created successfully!"
    }
}

```

```

    }
  } else {
    puts "$providerName already exists!"
  }

  return $provider
}

#-----
# createDB2orCloudscapeDatasource - Create a new Datasource if one
#       with the same name does not exist at the specified
#       scope. Otherwise, return the existing Datasource.
#-----
proc createDB2orCloudscapeDatasource {datasourceName jndiName stmtCacheSz provider providerType
authAliasName description scope dbName optArgs} {
  #   datasourceName      - Datasource name
  #   jndiName             - JNDI name
  #   stmtCacheSz         - Statement Cache Size
  #   provider             - provider
  #   providerType         - provider type (db2|db2cli|cloudscape)
  #   authAliasName        - JAAS authentication alias name
  #   description          - description
  #   scope                - scope
  #   dbName               - dbName (for DB2 and Cloudscape)
  #   optArgs[0] (jccType) - JDBC driver type (2|4)
  #   optArgs[1] (hostname) - database hostname (for type 4 driver)
  #   optArgs[2] (port)    - port number (for type 4 driver)

  set jccType [lindex $optArgs 0]
  set hostname [lindex $optArgs 1]
  set port [lindex $optArgs 2]

  global AdminConfig

  # Connection pool properties

  set connectionTimeout 1800
  set maxConnections 50
  set minConnections 1
  set reapTime 180
  set unusedTimeout 1800
  set agedTimeout 0

  if {${providerType} == "db2"} {
    set datasourceHelperClassname "com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper"
  } elseif {${providerType} == "db2cli"} {
    set datasourceHelperClassname "com.ibm.websphere.rsadapter.DB2DataStoreHelper"
  } elseif {${providerType} == "cloudscape"} {
    set datasourceHelperClassname "com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper"
  }

  set name [getName $scope]
  set stIndex [expr [string first "|" $scope] + 1]
  set endIndex [expr [string first "." $scope] - 1]
  set type [string range $scope $stIndex $endIndex]

  if {${type} == "cell"} {
    set radapter [AdminConfig getid "/Cell:$name/J2CResourceAdapter:WebSphere Relational Resource
Adapter/"]
  } elseif {${type} == "node"} {

```

```

        set radapter [$AdminConfig getid "/Node:$name/J2CResourceAdapter:WebSphere Relational Resource
Adapter/"]
    } elseif {{type} == "server"} {
        set radapter [$AdminConfig getid "/Server:$name/J2CResourceAdapter:WebSphere Relational Resource
Adapter/"]
    }

    puts " "
    puts "Creating DataSource ${datasourceName}..."

    # Check if the DataSource already exists

    foreach item [$AdminConfig list DataSource $scope] {
        set tmpProvider [$AdminConfig showAttribute $item provider]
        if {[string first $datasourceName $item] >= 0 && [string first $provider $tmpProvider] >= 0} {
            set datasource $item
            break
        }
        if {[string first $datasourceName $item] >= 0} {
            puts "${datasourceName} already exists in another JDBC Provider!"
            puts "Please rename or delete the existing DataSource before proceeding."
            exit
        }
    }

    # If DataSource does not yet exists, create a new one

    if {[info exists datasource]} {
        puts "  Datasource Name:      ${datasourceName}"
        puts "  JNDI Name:                  ${jndiName}"
        puts "  Statement Cache Size:         ${stmtCacheSz}"
        puts "  Database Name:                ${dbName}"
        if {$providerType == "db2"} {
            puts "  JDBC Driver Type:             ${jccType}"
            if {$jccType == 4} {
                puts "  Hostname:                      ${hostname}"
                puts "  Port Number:                   ${port}"
            }
        }
    }

    if {!($providerType == "cloudscape")} {
        set attrs [subst {{name $datasourceName} {description "$description"} {jndiName $jndiName}
{statementCacheSize $stmtCacheSz} {authDataAlias $authAliasName} {datasourceHelperClassname
$datasourceHelperClassname} {authDataAlias $authAliasName} {xaRecoveryAuthAlias "\""} {providerType
"[$AdminConfig showAttribute $provider providerType]"}]]
    } else {
        set attrs [subst {{name $datasourceName} {description "$description"} {jndiName $jndiName}
{statementCacheSize $stmtCacheSz} {datasourceHelperClassname $datasourceHelperClassname}
{xaRecoveryAuthAlias "\""} {providerType "[$AdminConfig showAttribute $provider providerType]"}]]
    }

    if {[catch {set datasource [$AdminConfig create DataSource $provider $attrs]} result]} {
        puts "WSADMIN EXCEPTION: ${result}"
        puts "Terminating due to exception!"
        exit
    } else {
        #Create the datasource properties...

        puts ""
        puts "  Creating Datasource properties..."
    }

```

```

set propSet    [$AdminConfig create J2EEResourcePropertySet $datasource {}]
set attrs [subst {{name databaseName} {type java.lang.String} {value $dbName}}}
$AdminConfig create J2EEResourceProperty $propSet $attrs

if {{providerType} == "db2"} {
  set attrs [subst {{name driverType} {type java.lang.Integer} {value $jccType}}}
  $AdminConfig create J2EEResourceProperty $propSet $attrs

  if {{jccType} == 4} {
    set attrs [subst {{name serverName} {type java.lang.String} {value $hostname}}}
    $AdminConfig create J2EEResourceProperty $propSet $attrs
    set attrs [subst {{name portNumber} {type java.lang.Integer} {value $port}}}
    $AdminConfig create J2EEResourceProperty $propSet $attrs
  }
}

#Create the connection pool object...
puts "  Creating Connection Pool using defaults..."
set attrs [subst {{connectionTimeout $connectionTimeout} {maxConnections $maxConnections}
{minConnections $minConnections} {reapTime $reapTime} {unusedTimeout $unusedTimeout} {agedTimeout
$agedTimeout}}]
$AdminConfig create ConnectionPool $datasource $attrs

#Create the connection factory
puts "  Creating Connection Factory..."
set cfName $datasourceName
append cfName "_CF"

if {{!(providerType == "cloudscape")}} {
  set attrs [subst {{name ${cfName}} {authMechanismPreference BASIC_PASSWORD} {cmpDatasource
$datasource} {authDataAlias $authAliasName}}}
} else {
  set attrs [subst {{name ${cfName}} {authMechanismPreference BASIC_PASSWORD} {cmpDatasource
$datasource}}}
}

$AdminConfig create CMPConnectorFactory $radapter $attrs

puts "${datasourceName} created successfully!"
}
} else {
  puts "${datasourceName} already exists in current JDBC Provider!"
}

return $datasource
}

#-----
# getName - Return the base name of the config object.
#-----
proc getName {args} {
  # arg[0] - object id

  set id [lindex $args 0]
  set endIndex [expr [string first "(" $id] - 1]

  return [string range $id 0 $endIndex]
}

```



```

}

# Remove everything the sample setup
# Assuming nobody changed the script file
# Utility proc mainly for test
#
# Typical invocation
# cleanup RuntimeDocumentMgmt DB2JDBCProv1 DB2AuthAlias
#
proc cleanup { {appName RuntimeDocumentMgmt} {jdbcProviderName DB2JDBCProv1} {jaasAliasName DB2AuthAlias}}
{
    global AdminApp
    global AdminConfig
    if {[catch {
        stopApp $appName
    } result]} { ;#ignore it }
    if {[catch {
        $AdminApp uninstall $appName
    } result]} { ;#ignore it }

    if {[catch {
        $AdminConfig remove [$AdminConfig getid /JDBCProvider:$jdbcProviderName/]
    } result]} { ;#ignore it }

    if {[catch {
        foreach i [$AdminConfig list JAASAuthData] {
            if { [$AdminConfig showAttribute $i alias] == $jaasAliasName } {
                $AdminConfig remove $i
            }
        }
    } result]} { ;#ignore it }

    $AdminConfig save
}

```

The WebSphereScript.jacl script file

Example A-10 shows the source of the WebSphereScript script.

Example: A-10 Source of WebSphereScript

```
# %I% %W% %G% %U%
# *****
# *****
# ***** Main routine *****
# ***** Get input values, verify all required values exist *****
# *****
# *****

puts " "
puts "Beginning of Script main"
puts "The number of passed arguments = $argc "

#-----
# This Script requires the following input parameters:
# 1 - directory where the scripts are located
# 2 - fully qualified name of response file
#-----
if {$argc != 2} {
    puts " "
    puts " This Script requires the following input parameters:"
    puts " 1 - the scripts directory"
    puts " 2 - fully qualified name of properties file"
    puts " "
    exit
} else {
    puts "Scripts dir = [lindex $argv 0] "
    puts "Properties File = [lindex $argv 1] "
}

#-----
# Get the input parms
#-----
set scriptsDir [lindex $argv 0]
set respFile [lindex $argv 1]

#-----
# Set source to point to the WebSphereConfigProcs
#-----
set setupScript [eval file join "$scriptsDir SetupProcs.jacl"]
set WasProcsScript [eval file join "$scriptsDir WebSphereConfigProcs.jacl"]

source $setupScript
source $WasProcsScript

#-----
# Use the Connected Node
#-----
global baseNode
set baseNode [$AdminControl getNode];# Get the one and only connected node
puts "base Node = $baseNode"
set baseCell [$AdminControl getCell];# Get the one and only connected cell
puts "base Cell = $baseCell"

#-----
```

```

# Get the variables values from the response file
#-----

set props [loadProperties $respFile];# load properties from the response file

set DB2Deploy          "DB2UDB_V82"

#-----
# JMS (Messaging) Config Parameters
#-----

#set reliability        "ASSURED_PERSISTENT"
set reliability         "EXPRESS_NONPERSISTENT"

#set deliveryMode       "Persistent"
set deliveryMode        "NonPersistent"

# Queue/Topic Names
set brokerSIBDest       "TradeBrokerJSD"
set topicSpace          "Trade.Topic.Space"
set brokerJMSQCF        "TradeBrokerQCF"
set streamerJMSTCF      "TradeStreamerTCF"
set brokerQueue         "TradeBrokerQueue"
set streamerTopic       "TradeStreamerTopic"
set brokerMDB           "TradeBrokerMDB"
set streamerMDB         "TradeStreamerMDB"

# *****
# *****
# ***** Start of Configuratin Steps *****
# *****
# *****

#-----
# Set scope - id of scope (cell, node, or server)
#-----

set nodeScope [$AdminConfig getid /Node:$baseNode/]
set serverScope [$AdminConfig getid /Node:$baseNode/Server:$wasServerName]

#-----
# Create a JAAS authorization alias ( required for V5 DataSources)
#-----

if {[catch {
    createJAASDataAuth $authAliasName $db2UserId $db2Password $authAliasDesc
  } result]} {
    # this is an unrecoverable error. Stack should have enough information to rectify
    dumpStack $result
    exit
}

#-----
# Create DB2 JDBCProvider
#-----

if {[catch {

```

```

# createJDBCProviderUsingTemplate $JDBCProviderName $JDBCclasspath $baseNode $wasServerName
set provider [createJDBCProvider db2 $JDBCclasspath "true" $nodeScope]

} result]] {
# this is an unrecoverable error. Stack should have enough information to rectify
dumpStack $result
exit
}

#-----
# Create DB2 DataSource
#-----
set authMech "BASIC_PASSWORD"
set dsHelperClass "com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper"

if [[catch {
#   createDB2DataSource $baseNode $wasServerName $JDBCProviderName $datasourceName $databaseName
#   dsHelperClass $authAliasName $authMech $datasourceCategory $datasourceDesc
    set jdbcProvid [$AdminConfig getid /JDBCProvider:$JDBCProviderName/]
    set jccParms "4 localhost 50000"

    createDB2orCloudscapeDataSource $datasourceName "jdbc/${datasourceName}" 60 $provider db2
    $authAliasName "Trade6 DataSource" $nodeScope $databaseName $jccParms
} result]] {
# this is an unrecoverable error. Stack should have enough information to rectify
dumpStack $result
exit
}

#-----
# Create the JMS config objects
#-----

puts ""
puts "-----"
puts " Configuring JMS Resources"
puts "-----"

createJAASDataAuth $JMSAuthAliasName $DefaultOSUser $DefaultOSPasswd $JMSAuthAliasDesc

set SIBusName [createSIBus $baseNode $JMSAuthAliasName]
set target [subst {$baseNode $wasServerName}]
addSIBusMember $SIBusName "true" "dummy" $target

# Create the Trade Broker Queue and Trade TopicSpace Destinations

createSIBDestination $SIBusName $brokerSIBDest "Queue" $reliability $target
createSIBDestination $SIBusName $topicSpace "TopicSpace" $reliability [subst {}]

createJMSConnectionFactory $SIBusName $brokerJMSQCF "Queue" "jms/$brokerJMSQCF" $JMSAuthAliasName
$nodeScope
createJMSConnectionFactory $SIBusName $streamerJMSTCF "Topic" "jms/$streamerJMSTCF" $JMSAuthAliasName
$nodeScope

createJMSQueue $brokerQueue "jms/$brokerQueue" $brokerSIBDest $deliveryMode $nodeScope
createJMSTopic $streamerTopic "jms/$streamerTopic" $topicSpace $deliveryMode $nodeScope

```

```

    createMDBActivationSpec $brokerMDB "eis/$brokerMDB" $SIBusName "jms/$brokerQueue" "javax.jms.Queue"
$JMSAuthAliasName $nodeScope
    createMDBActivationSpec $streamerMDB "eis/$streamerMDB" $SIBusName "jms/$streamerTopic"
"javax.jms.Topic" $JMSAuthAliasName $nodeScope

    puts ""
    puts "-----"
    puts " JMS Resource Configuration Completed!!!"
    puts "-----"

#-----
# Install the Application
# If the application install file is an EAR file then use installEar
# If the application install file is a WAR file then use installWar and include the contextroot
#-----
puts "Install EAR File"
# the ear file was placed into the WAS installable apps directory
set qualifiedAppInstallFile [eval file join "$installableAppsDir $appInstallFile"]
if {[catch {
    # installEar $wasServerName $qualifiedAppInstallFile $appName $contextRoot
    installApp $appName $qualifiedAppInstallFile "true" "true" "true" "true" $DB2Deploy $target
    } result]} {
    # this is an unrecoverable error. Stack should have enough information to rectify
    dumpStack $result
    exit
}

# By default the application will only have been added to server1. Must also add it to webserver1 for it
# to be accessible through the http server

# Define the WAS server in the format expected by the MapModulesToServers option
set server1 WebSphere:cell=$baseCell,node=$baseNode,server=$wasServerName
puts " server1 = $server1 "
# Define the HTTP server in the format expected by the MapModulesToServers option
set webserver1 WebSphere:cell=$baseCell,node=$webServerNode,server=$webServerName
puts " webserver1 = $webserver1 "

# Append the two together with a + delimiter
append server1 +$webserver1

set parms "-MapModulesToServers {{{\"TradeWeb\" tradeWeb.war,WEB-INF/web.xml $server1}}}"
set parms1 [subst {$parms}]
$AdminApp edit Trade $parms1

#-----
# IMPORTANT
# Save the configuration changes made
#-----
saveConfiguration

# Start the application to make the URL active
# Note that an invocation of this script with the "-conntype NONE" parameter will
# cause this procedure to generate an error
startApp $appName $wasServerName

```

```

# special Linux section
if {[string first "Linux" $env(os.name)] == 0} {
    # Have to add DB2 environment variables to the server process definition
    # This is done to avoid having to source the db2profile
    # we want to strip the path down to db2 home
    # e.g. /opt/IBM/db2/V8.1/java/db2java.zip becomes /opt/IBM/db2/V8.1
    secureDB2 [file dirname [file dirname $JDBCclasspath]] $db2instance
}

# *****
# End Main routine
# *****

```

The Table.ddl file

Example A-11 shows the source of the Table.ddl file.

Example: A-11 Table.ddl

```

DROP TABLE HOLDINGEJB;
DROP TABLE ACCOUNTPROFILEEJB;
DROP TABLE QUOTEEJB;
DROP TABLE KEYGENEJB;
DROP TABLE ACCOUNTEJB;
DROP TABLE ORDEREJB;

CREATE TABLE HOLDINGEJB
(PURCHASEPRICE DECIMAL(10, 2),
HOLDINGID INTEGER NOT NULL,
QUANTITY DOUBLE NOT NULL,
PURCHASEDATE TIMESTAMP,
ACCOUNT_ACCOUNTID INTEGER,
QUOTE_SYMBOL VARCHAR(250));

ALTER TABLE HOLDINGEJB
ADD CONSTRAINT PK_HOLDINGEJB PRIMARY KEY (HOLDINGID);

CREATE TABLE ACCOUNTPROFILEEJB
(ADDRESS VARCHAR(250),
PASSWORD VARCHAR(250),
USERID VARCHAR(250) NOT NULL,
EMAIL VARCHAR(250),
CREDITCARD VARCHAR(250),
FULLNAME VARCHAR(250));

ALTER TABLE ACCOUNTPROFILEEJB
ADD CONSTRAINT PK_ACCOUNTPROFILE2 PRIMARY KEY (USERID);

CREATE TABLE QUOTEEJB
(LOW DECIMAL(10, 2),
OPEN1 DECIMAL(10, 2),
VOLUME DOUBLE NOT NULL,
PRICE DECIMAL(10, 2),
HIGH DECIMAL(10, 2),
COMPANYNAME VARCHAR(250),
SYMBOL VARCHAR(250) NOT NULL,
CHANGE1 DOUBLE NOT NULL);

```

```

ALTER TABLE QUOTEEJB
  ADD CONSTRAINT PK_QUOTEEJB PRIMARY KEY (SYMBOL);

CREATE TABLE KEYGENEJB
  (KEYVAL INTEGER NOT NULL,
   KEYNAME VARCHAR(250) NOT NULL);

ALTER TABLE KEYGENEJB
  ADD CONSTRAINT PK_KEYGENEJB PRIMARY KEY (KEYNAME);

CREATE TABLE ACCOUNTEJB
  (CREATIONDATE TIMESTAMP,
   OPENBALANCE DECIMAL(10, 2),
   LOGOUTCOUNT INTEGER NOT NULL,
   BALANCE DECIMAL(10, 2),
   ACCOUNTID INTEGER NOT NULL,
   LASTLOGIN TIMESTAMP,
   LOGINCOUNT INTEGER NOT NULL,
   PROFILE_USERID VARCHAR(250));

ALTER TABLE ACCOUNTEJB
  ADD CONSTRAINT PK_ACCOUNTEJB PRIMARY KEY (ACCOUNTID);

CREATE TABLE ORDEREJB
  (ORDERFEE DECIMAL(10, 2),
   COMPLETIONDATE TIMESTAMP,
   ORDERTYPE VARCHAR(250),
   ORDERSTATUS VARCHAR(250),
   PRICE DECIMAL(10, 2),
   QUANTITY DOUBLE NOT NULL,
   OPENDATE TIMESTAMP,
   ORDERID INTEGER NOT NULL,
   ACCOUNT_ACCOUNTID INTEGER,
   QUOTE_SYMBOL VARCHAR(250),
   HOLDING_HOLDINGID INTEGER);

ALTER TABLE ORDEREJB
  ADD CONSTRAINT PK_ORDEREJB PRIMARY KEY (ORDERID);

ALTER TABLE HOLDINGEJB VOLATILE;
ALTER TABLE ACCOUNTPROFILEEJB VOLATILE;
ALTER TABLE QUOTEEJB VOLATILE;
ALTER TABLE KEYGENEJB VOLATILE;
ALTER TABLE ACCOUNTEJB VOLATILE;
ALTER TABLE ORDEREJB VOLATILE;

CREATE INDEX a.profile_userid on accountejb(profile_userid);
CREATE INDEX h.account_accountid on holdingejb(account_accountid);
CREATE INDEX o.account_accountid on orderejb(account_accountid);
CREATE INDEX o.holding_holdingid on orderejb(holding_holdingid);
CREATE INDEX o.orderstatus on orderejb(orderstatus);
CREATE INDEX o.ordertype on orderejb(ordertype);

```

The DB2Script.bat file

Example A-12 shows the source for the DB2Script.bat file.

Example: A-12 DB2Script.bat

```
@echo off
goto afterProlog
REM *****
REM The script expects some parameters:
REM      %1 - the database name
REM      %2 - the full path to the script directory
REM      %3 - db2 userid
REM      %4 - db2 password
REM      Ignores any other parameters

REM Note: this Batch file must be run in a DB2Cmd window.
REM -----
REM Create Database
REM -----
:afterProlog

if not .%2 == . goto run

:help
echo.
echo DB2Script: this script requires 2 parameters:
echo      1) the database name
echo      2) the full path to the scripts directory
echo      3) DB2 userid
echo      4) DB2 password
echo.
goto end

REM Strip out the double quotes if any exist
:run
set dbName=%1
set dbName=%dbName:"=%
set ScriptDirectory=%2
set ScriptDirectory=%ScriptDirectory:"=%
set db2userid=%3
set db2userid=%db2userid:"=%
set db2password=%4
set db2password=%db2password:"=%

echo Create the trade database

echo --
echo -- cataloging is necessary to recover from a previous installation's left over
database
echo -- please ignore any error messages emitted
echo -- we also do an uncatalog because DB2 does not check to see whether the DB exists, it
blindly catalogs it.
echo -- so that would cause the rest of the script to fail.
echo --
db2 catalog database %dbName%
db2 drop database %dbName%
db2 uncatalog database %dbName%
```



```

db2 create database %dbName%

db2 connect to %dbName% user %db2userid% using %db2password%

db2 -tvf "%ScriptDirectory%\Table.ddl"

db2 disconnect all
db2 update db config for %dbName% using logfilsiz 1000
db2 update db cfg for %dbName% using maxappls 100
db2stop force
db2start

:end

```

The SetupProcs.jacl script file

Example A-13 shows the source of the SetupProcs.jacl file.

Example: A-13 SetupProcs.jacl

```

# %I% %W% %G% %U%
#-----
# This script is used to set the values specified in
# the application response file
# It will set the variables required by the jacl scripts.
# Note that this may not include all of the
# variables specified in the response file.
#
# When adding a variable
# 1 - The variable must be defined as global
# 2 - set the variable value from the response file
# 3 - "puts" the variable and value
#    so it shows up in the log file
#-----

proc loadProperties {propFileName} {
    global env
    puts "SetupProcs - Loading properties"
    java::import java.io.FileInputStream
    java::import java.util.Properties

#-----
# Load the response file as Properties
#-----
    set props [java::new Properties]
    set fileStream [java::new FileInputStream $propFileName]
    $props load $fileStream

#-----
# The scripts directory is used to find the scripts included in the user program filelist
# as specified in the application.xml
#-----
    global scriptsDir
    set scriptsDir [string trim [$props getProperty DOCMGMT.scriptsDir ]]
    puts " scriptsDir = $scriptsDir "

#-----
# Set the WAS configuration variables

```

```

#-----
global wasServerName installableAppsDir appInstallFile contextRoot JDBCProviderName
JDBCclasspath
global datasourceName datasourceDesc datasourceCategory
global authAliasName authAliasDesc
global appName

global JMSauthAliasName JMSauthAliasDesc
global DefaultOSUser DefaultOSPasswd

set wasServerName [string trim [$props getProperty WAS.serverName ]]
puts " wasServerName = $wasServerName "
set installableAppsDir [string trim [$props getProperty WAS.installableAppsDir ]]
puts " installableAppsDir = $installableAppsDir "
set appInstallFile [string trim [$props getProperty WAS.appInstallFile ]]
puts " appInstallFile = $appInstallFile "
set JDBCProviderName [string trim [$props getProperty WAS.JDBCProviderName ]]
puts " JDBCProviderName = $JDBCProviderName "
set JDBCclasspath [string trim [$props getProperty WAS.JDBCclasspath ]]
puts " JDBCclasspath = $JDBCclasspath "
set datasourceName [string trim [$props getProperty WAS.datasourceName ]]
puts " datasourceName = $datasourceName "
set datasourceDesc [string trim [$props getProperty WAS.datasourceDesc ]]
puts " datasourceDesc = $datasourceDesc "
set datasourceCategory [string trim [$props getProperty WAS.datasourceCategory ]]
puts " datasourceCategory = $datasourceCategory "
set authAliasName [string trim [$props getProperty WAS.authAliasName ]]
puts " authAliasName = $authAliasName "
set authAliasDesc [string trim [$props getProperty WAS.authAliasDesc ]]
puts " authAliasDesc = $authAliasDesc "
set appName [string trim [$props getProperty WAS.appName ]]
puts " appName = $appName "
set contextRoot [string trim [$props getProperty WAS.contextRoot ]]
puts " contextRoot = $contextRoot "

set JMSauthAliasName [string trim [$props getProperty WAS.JMSauthAliasName ]]
puts " JMSauthAliasName = $JMSauthAliasName "
set JMSauthAliasDesc [string trim [$props getProperty WAS.JMSauthAliasDesc ]]
puts " JMSauthAliasDesc = $JMSauthAliasDesc "

set DefaultOSUser [string trim [$props getProperty WAS.DefaultOSUser ]]
puts " DefaultOSUser = $DefaultOSUser "
set DefaultOSPasswd [string trim [$props getProperty WAS.DefaultOSPasswd ]]
puts " DefaultOSPasswd = $DefaultOSPasswd "

#-----
# Set the IHS configuration variables
#-----
global webServerName webServerNode

set webServerName [string trim [$props getProperty IHS.serverName ]]
puts " webServerName = $webServerName "
set webServerNode [string trim [$props getProperty IHS.nodeName ]]
puts " webServerNode = $webServerNode "

#-----
# Set the DB2 variables
#-----
global databaseName db2instance db2UserId db2Password

```

```

    set databaseName [string trim [$props getProperty DB2.databaseName ]]
    puts " databaseName = $databaseName "
    set db2instance [string trim [$props getProperty DB2.db2instance ]]
    puts " db2instance = $db2instance "
    set db2UserId [string trim [$props getProperty DB2.AdminID ]]
    puts " db2UserId = $db2UserId "
    set db2Password [string trim [$props getProperty DB2.AdminPW ]]
    puts " db2Password = ***** "

    puts " "
    puts "End SetupProcs Script "
    puts " "
}

```

The Trade.prop file

Example A-14 shows the source of the Trade.prop file.

Example: A-14 Trade.prop

```

#-----
#
# Trade Properties File
#
# HINTS:
#   - Set path names using shortname, e.g., "Program Files" is "progra~1"
#   - Use forward slash in directory entries
#
# Note that some of these values are set in Main
#-----

#   The name of the WebSphere configuration script to be run
TRADE.WASscript=WebSphereScript.jacl

#   The name of the DB2 script to be run
TRADE.DB2script=DB2Script.bat

#   The scripts are part of the userPrograms fileList in the application.xml.
#   The scriptsDir is the directory under which the scripts were placed, as
#   specified in the application.xml.
TRADE.scriptsDir=Trade_ScriptsDir

#   Variables used for WAS
WAS.version=6.0.0.0
WAS.serverName=server1
WAS.profile=default
#   WAS installable apps directory is set in the Main
WAS.installableAppsDir=

# -----
#   Variables used for HTTP server configuration steps
# -----

IHS.serverName=webserver1
IHS.nodeName=webserver1_node

# -----

```

```

# Variables used for WAS configuration steps
# -----

# The name of the application to be installed
WAS.appName=Trade

# the install EAR or WAR file
WAS.appInstallFile=trade.ear

# the contextRoot -- used if the application is installed using a war file
WAS.contextRoot=trade

# the JDBC classpath will be set in the Main --
c:/progra~1/IBM/SQLLIB/java/db2java.zip
WAS.JDBCclasspath=

# the JDBCProvider name to be created
WAS.JDBCProviderName=TradeDB2Provider

# the DataSource to be created
WAS.datasourceName=TradeDataSource
WAS.datasourceDesc="This datasource is used by the Trade Application"
WAS.datasourceCategory="Trade Application"

# the J2C Authorization entry to be created
WAS.authAliasName=TradeDataSourceAuthData
WAS.authAliasDesc="JAASDataAuth for the Trade database"

# the J2C Authorization entry to be created for JMS resource
WAS.JMSauthAliasName=TradeOSUserIDAAuthData
WAS.JMSauthAliasDesc="JAASDataAuth for the Trade JMS resources"

# the Local OS userid and password
WAS.DefaultOSUser=LocalOSUserID
WAS.DefaultOSPasswd=password


# the DB2 database name and instance
DB2.databaseName=tradedb
DB2.db2instance=DB2
# the DB2 Administrator userid and password
DB2.AdminID=
DB2.AdminPW=

# The name of the IHS service, in quotes, to be started using the net command,
# e.g., net start "IBM HTTP Server 6.0"
# If a web server service has not been defined on the system, do not provide a value
IHS.webServerService="IBM HTTP Server
6.0"

```

Source code for Trade6 user programs and script files for Linux on POWER

This appendix provides the source codes used in the Trade6 solution example required for Linux on POWER. The following programs and files are included:

- ▶ application.xml
- ▶ solution.xml
- ▶ TradeLnxMain.java
- ▶ TradeLnxPDC.java
- ▶ TradeLnxCommon.java
- ▶ TradeNLSKeys.java
- ▶ TradeMessagesNLS.java
- ▶ TradeMessagesNLS_en.java
- ▶ CheckAppInstall.jacl
- ▶ WebSphereConfigProcs.jacl
- ▶ WebSphereScript.jacl
- ▶ DB2Script.bat
- ▶ Table.ddl
- ▶ SetupProcs.jacl
- ▶ Trade.prop file

The application.xml file

Example B-1 shows the source of the application.xml file.

Example: B-1 Source of application.xml

```
<?xml version="1.0" ?>

<iru:application
  id="Trade6Linuxi5"
  xmlns:iru="http://www.ibm.com/xmlns/prod/iru/application"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/xmlns/prod/iru/application IRU_application.xsd">

  <applicationInformation
    installTime="120"
    version="1.0">
    <name>Trade6Linuxi5</name>
    <operatingSystems>
      <operatingSystem>LinuxOnPOWER</operatingSystem>
    </operatingSystems>
    <providerName>IBM</providerName>
  </applicationInformation>

  <translationLanguages default="english">
    <language>english</language>
  </translationLanguages>

  <fileLists>
    <fileList id="softwareimagefiles">
      <file>trade.ear</file>
    </fileList>
    <fileList
      id="userprogramfiles"
      userPrograms="true">
      <file>com/trade/TradeLnxCommon.class</file>
      <file>com/trade/TradeLnxMain.class</file>
      <file>com/trade/TradeLnxPDC.class</file>
      <file>com/trade/TradeMessagesNLS.class</file>
      <file>com/trade/TradeMessagesNLS_en.class</file>
      <file>com/trade/TradeNLSKeys.class</file>
      <file>Trade_ScriptsDir/CheckAppInstall.jacl</file>
      <file>Trade_ScriptsDir/DB2Script.sh</file>
      <file>Trade_ScriptsDir/IRU_EditMap.jacl</file>
      <file>Trade_ScriptsDir/IRU_ViewApp.jacl</file>
      <file>Trade_ScriptsDir/SetupProcs.jacl</file>
      <file>Trade_ScriptsDir/Table.ddl</file>
      <file>Trade_ScriptsDir/WebSphereConfigProcs.jacl</file>
      <file>Trade_ScriptsDir/WebSphereScript.jacl</file>
    </fileList>
  </fileLists>

  <preDeploymentChecker
    logFile="Trade6Linuxi5PDC.log"
    programName="com.trade.TradeLnxPDC"
    responseFile="Trade.prop"
```

```

        successType="returnCode"
        type="java">
            <arguments>
                <argument responseFile="true" />
            </arguments>
        </preDeploymentChecker>

<mainProgram
    logFile="Trade6Linuxi5Main.log"
    programName="com.trade.TradeLnMain"
    responseFile="Trade.prop"
    successType="returnCode"
    type="java">
        <arguments>
            <argument responseFile="true" />
        </arguments>
    </mainProgram>

<variables>
    <stringVariable
        maxLength="20"
        minLength="2"
        name="DB2UserId"
        required="true">
        <labelText>DB2 Admin user</labelText>
        <propertiesAssociations>
            <propertiesAssociation keyword="DB2UserId" />
        </propertiesAssociations>
        <inputValidation>
            <valid>
                <characters>@#$%&'()*+,-./:;<=;>?[]^_`{|}~0123456789</characters>
            </valid>
        </inputValidation>
        <defaultData>db2inst</defaultData>
    </stringVariable>
    <passwordVariable
        maxLength="127"
        minLength="6"
        name="DB2UserPassword"
        required="true">
        <labelText>DB2 Admin Password</labelText>
        <propertiesAssociations>
            <propertiesAssociation keyword="DB2UserPassword" />
        </propertiesAssociations>
        <inputValidation>
            <valid>
                <characters>@#$%&'()*+,-./:;<=;>?[]^_`{|}~0123456789</characters>
            </valid>
        </inputValidation>
        <defaultData>db2inst</defaultData>
    </passwordVariable>
</variables>

</iru:application>

```

The solution.sxml file

Example B-2 shows the source of the solution.sxml file.

Example: B-2 Source of solution.sxml

```
<?xml version="1.0" ?>

<iru:solution
  id="Trade6Linuxi5Solution"
  xmlns:iru="http://www.ibm.com/xmlns/prod/iru/solution"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/xmlns/prod/iru/solution IRU_solution.xsd">

  <solutionInformation>
    <title>Trade6Linuxi5Solution</title>
  </solutionInformation>

  <translationLanguages default="english">
    <language>english</language>
  </translationLanguages>

  <tasks>
    <taskGroup>
      <taskGroupTitle>Trade6 Application</taskGroupTitle>
      <taskGroupPrompt>Trade6 Application</taskGroupPrompt>
      <installTask
        isOptional="true"
        operatingSystem="LinuxOnPOWER">
        <description>Trade6 Install</description>
        <applications>
          <application fileName="Trade6Linuxi5_powerlinux.ser" />
        </applications>
      </installTask>
    </taskGroup>
  </tasks>

</iru:solution>
```

The TradeLnxMain.java program

Example B-3 is a sample TradeLnxMain.java file which was used for deploying the Trade6 application.

Example: B-3 TradeLnxMain.java

```
package com.trade;
/**
 * @(#)SampleLnxMain.java
 *
 * Licensed Materials - Property of IBM
 *
 * 5724-F71 5724-J10
 */
```



```

* (C) Copyright IBM Corp. 2004, 2005
*
* US Government Users Restricted Rights - Use, duplication or disclosure
* restricted by GSA ADP Schedule Contract with IBM Corp.
*
* This is sample code made available for use in accordance with terms set
* forth in the license agreement document for the IBM Express Runtime.
*/

import com.trade.TradeNLSKeys;
import com.ibm.jsdt.support.*;

/**
 * This class installs the application contained in the passed war file
 */
public class TradeLnMain extends SupportLinuxBase {
    private static final String copyright0="Licensed Materials - Property of IBM";
    private static final String copyright1="5724-F71 5724-J10";
    private static final String copyright2="(C) Copyright IBM Corp. 2004, 2005 All Rights
Reserved.";
    private static final String copyright3="US Government Users Restricted Rights - Use,
duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.";

    private SupportLinuxHelper ivHelper = null;
    private String ivCommandResult = null;
    private String ivWasProfileName = null;
    private String ivWasProfileBinDir = null;
    private String ivInstallableAppsDir = null;
    private String ivDB2InstallDir = null;
    private String ivDbName = null;
    private String ivWasServerName = null;
    private String ivWasVpdUid = null;
    private String ivIhsVpdUid = null;
    private String ivIhsServerName = null;
    private String ivIHSBinDir = null;
    private String ivWasInstallDir = null;
    private String ivWasScript = null;
    private String ivDB2Version = null;
    private String ivDB2Script = null;
    private String ivDB2InstanceId = null;
    private String ivDocsDir = null;
    private String ivScriptsDir = null;
    private String ivAppFile = null;
    private String ivAppName = null;
    private String ivDB2UserId = null;
    private String ivDB2PassWord = null;

    /**
     * Initializes the support framework and checks the right number of arguments were
     passed.
     * @param args
     * @return SUCCESS if the right number of parameters were passed, otherwise FAILURE
     */
    public int init(String [] args)
    {
        int rc = SUCCESS;
        ivHelper = getLinuxHelper();
        setMainResources(TradeLnCommon.Trade_MESSAGES);

        if (args.length != 1)

```

```

        {
            setMessage(getResourceString(TradeNLSKeys.BAD_NUMBER_PGM_ARGS, new String[] {
                "TradeLnMain",
                "1",
                Integer.toString(args.length) }));
            ivHelper.log(this);
            rc = FAILURE;
        }
    else
    {
        // The properties file is in the same format as a response file
        setResponseFileName(args[0]);
    }

    return rc;
}
/**
 * Install and configure the application
 * This will
 * - get the DB2 Instance userid and password entered by the user (from the support
framework properties file)
 * - get the values for other required variables from the application properties file
 * - get the WAS directories, both the install dir and the bin dir
 * - get the DB2 install directory
 * - set the jdbc classpath (based on the DB2 install directory)
 * - copy the EAR or WAR file to the WAS installableApps directory
 * - copy the Trade documents to the specified directory
 * - run the DB2 script to create and populate the database
 * - run the WAS script to do the WAS configuration
 * - generate the HTTP plug-in
 * - restart the HTTP Server
 */
private int install()
{
    int rc = SUCCESS;

    // get the values set in the properties file, properties file is mandatory
    if (getResponseFileName() == null || getResponseFileName().trim().equals("")) {
        setMessage(getResourceString(TradeNLSKeys.NO_PROPERTIES_FILE));
        ivHelper.log(this);
        rc = FAILURE;
    } else {
        rc = getProperties();
    }

    // Determine if WAS is installed and get the WAS directories
    if (rc == SUCCESS) {
        rc = determineWasDir();
    }

    // Determine if DB2 is installed
    if (rc == SUCCESS) {
        rc = determineDB2Dir();
    }

    // Determine if IHS is installed
    if (rc == SUCCESS){
        rc = determineIHSDir();
    }
}

```

```

// Set values in the properties file
if (rc == SUCCESS) {
    setProperties();
}

if (rc == SUCCESS) {
    // copy the application EAR or WAR file into the "installable apps" directory
    rc = copyApp();
}

if (rc == SUCCESS) {
    // install the Trade files used in conjunction with the DB/2 database
    installTradeFiles();
    // run the DB/2 Script
    rc = runDB2Script();
}

if (rc == SUCCESS) {
    // Run the WAS Script
    rc = runWebSphereScript();
}

// Generate the HTTP Plugin
if (rc == SUCCESS) {
    rc = generateHTTPPlugin();
}

// Restart the WAS Server so the configuration changes will be in effect
if (rc == SUCCESS) {
    rc = restartWASServer();
}

// Restart the HTTP Server so the plugin will be enabled
if (rc == SUCCESS) {
    // if IHS specified in the application properties file -- restart it
    if (getIHSBinDir() != null && !getIHSBinDir().trim().equals("")) {
        rc = restartHTTPServer();
    } else {
        setMessage(getResourceString(TradeNLSKeys.NO_HTTPSERVER));
        ivHelper.log(this);
    }
}

return rc;
}

/**
 * Restart the WebSphere Application Server
 */
private int restartWASServer()
{
    int rc = SUCCESS;
    // stop the server, ignore the return code - the real check is on the start
    command.
    String [] stopcommand = {"/bin/sh", "-c", getWasProfileBinDir() + "stopServer.sh "
+ getServerName()};

    rc = invokeCommand(stopcommand);
    if (rc != SUCCESS) {
        // log an error if not successful

```

```

        setMessage(getResourceString(TradeNLSKeys.STOP_WASEXPRESS_FAIL));
        ivHelper.log(this);
    }

    // Start the server and check the results
    String [] startcommand = {"/bin/sh", "-c", getWasProfileBinDir() + "startServer.sh
" + getServerName()};
    rc = invokeCommand(startcommand);
    // Check the result of the start command
    if (rc != SUCCESS) {
        // log an error if not successful
        setMessage(getResourceString(TradeNLSKeys.START_WASEXPRESS_FAIL));
        ivHelper.log(this);
    }
    return rc;
}

/**
 * Get values from the properties files
 */
private int getProperties()
{
    int rc = SUCCESS;
    // Get the properties from the support framework Nsi Properties file
    setVariableName("DB2UserId");
    setDB2InstanceId(ivHelper.getIbmNsiPropValue(this));
    setVariableName("DB2UserPassword");
    setDB2PW(ivHelper.getIbmNsiPropValue(this));

    // Get the values specified in the application properties file
    // The application properties file name was set in main, this file is
    // in the same format as a response file therefore the support framework
    // getResponseFileValue and setResponseFileValue methods can be used

    //set the WAS UID used in the ISMP vpd.properties file, needed to get the install
directory
    setKey(TradeLnxCommon.WAS_VPD_UID);
    setWasVpdUid(ivHelper.getResponseFileValue(this));
    if (getWasVpdUid() == null || getWasVpdUid().trim().equals("")) {
        setMessage(getResourceString(TradeNLSKeys.NO_PROPERTY, TradeLnxCommon.WAS_VPD_UID));
        ivHelper.log(this);
        rc = FAILURE;
    }

    //set the DB2 version, as it appears in the rpm database. Needed to get the install
directory
    setKey(TradeLnxCommon.DB2_VERSION);
    setDB2Version(ivHelper.getResponseFileValue(this));
    if (getDB2Version() == null || getDB2Version().trim().equals("")) {
        setMessage(getResourceString(TradeNLSKeys.NO_PROPERTY, TradeLnxCommon.DB2_VERSION));
        ivHelper.log(this);
        rc = FAILURE;
    }

    //set the IHS UID used in the ISMP vpd.properties file, needed to get the install
directory
    setKey(TradeLnxCommon.IHS_VPD_UID);
    setIhsVpdUid(ivHelper.getResponseFileValue(this));
    if (getIhsVpdUid() == null || getIhsVpdUid().trim().equals("")) {
        setMessage(getResourceString(TradeNLSKeys.NO_PROPERTY, TradeLnxCommon.IHS_VPD_UID));

```

```

        ivHelper.log(this);
        rc = FAILURE;
    }

    setKey(TradeLnxCCommon.SERVER_NAME_KEY); // server name to use for the start
command
    setServerName(ivHelper.getResponseFileValue(this));
    if (getServerName() == null || getServerName().trim().equals("")) {
        setMessage(getResourceString(TradeNLSKeys.NO_PROPERTY,
TradeLnxCCommon.SERVER_NAME_KEY));
        ivHelper.log(this);
        rc = FAILURE;
    }

    setKey(TradeLnxCCommon.WAS_PROFILE_KEY); // WAS profile name
    setWasProfileName(ivHelper.getResponseFileValue(this));
    if (getWasProfileName() == null || getWasProfileName().trim().equals("")) {
        setMessage(getResourceString(TradeNLSKeys.NO_PROPERTY,
TradeLnxCCommon.WAS_PROFILE_KEY));
        ivHelper.log(this);
        rc = FAILURE;
    }

    setKey(TradeLnxCCommon.APP_NAME_KEY); // the name of the application to be installed,
it is required
    setAppName(ivHelper.getResponseFileValue(this));
    if (getAppName() == null || getAppName().trim().equals("")) {
        setMessage(getResourceString(TradeNLSKeys.NO_PROPERTY,
TradeLnxCCommon.APP_NAME_KEY));
        ivHelper.log(this);
        rc = FAILURE;
    }

    setKey(TradeLnxCCommon.IHS_SERVER_NAME_KEY); // the name of the IHS server, not
required
    setIhsServerName(ivHelper.getResponseFileValue(this));
    if (getIhsServerName() == null || getIhsServerName().trim().equals("")) {
        setMessage(getResourceString(TradeNLSKeys.NO_PROPERTY,
TradeLnxCCommon.IHS_SERVER_NAME_KEY));
        ivHelper.log(this);
    }

    setKey(TradeLnxCCommon.DB2_SCRIPT_KEY); // DB2 script
    setDB2Script(ivHelper.getResponseFileValue(this));

    setKey(TradeLnxCCommon.WAS_SCRIPT_KEY); // WAS script
    setWasScript(ivHelper.getResponseFileValue(this));

    setKey(TradeLnxCCommon.DOCS_DIR_KEY); // the directory where the application
documents are to be installed
    setDocsDir(ivHelper.getResponseFileValue(this));

    setKey(TradeLnxCCommon.SCRIPTS_DIR_KEY); // the directory within userPrograms where
the scripts are located
    setScriptsDir(ivHelper.getResponseFileValue(this));

    setKey(TradeLnxCCommon.DB_NAME_KEY); // the name of the database
    setDbName(ivHelper.getResponseFileValue(this));

    setKey(TradeLnxCCommon.APP_FILE_KEY); // the ear or war file to be installed

```

```

        setAppFile(ivHelper.getResponseFileValue(this));

        return rc;
    }

    /**
     * Set values in the application properties file that are needed by the jacl scripts
     */
    private void setProperties()
    {
        // set the JDBC Provider classpath in the properties file
        setKey(TradeLnxCCommon.JDBC_CLASSPATH_KEY);
        // the JDBC classpath is based on the DB2 install directory
        setKeyValue(getDB2InstallDir() + "java/db2jcc.jar");
        ivHelper.setResponseFileValue(this);

        // set the WAS installable apps directory in the properties file
        setKey(TradeLnxCCommon.WAS_INSTALLABLE_APPS_DIR_KEY);
        setKeyValue(getInstallableAppsDir());
        ivHelper.setResponseFileValue(this);

        // set the directory for the Trade app files
        setKey(TradeLnxCCommon.DOCS_DIR_KEY);
        setKeyValue(getDocsDir());
        ivHelper.setResponseFileValue(this);

        // include the contents of the updated properties file in the log
        ivHelper.logNewLine(this);
        setFileName(getResponseFileName());
        ivHelper.logAppendFile(this);
        ivHelper.logNewLine(this);

        // set the db2 instance user id and password
        setKey(TradeLnxCCommon.DB2_INSTANCE_ID_KEY);
        setKeyValue(getDB2InstanceId());
        ivHelper.setResponseFileValue(this);
        setKey(TradeLnxCCommon.DB2_INSTANCE_PW_KEY);
        setKeyValue(getDB2PW());
        ivHelper.setResponseFileValue(this);
    }

    /**
     * Copies the EAR or WAR to the WAS installableApps directory
     * Returns SUCCESS or FAILURE
     */
    private int copyApp()
    {
        int rc = FAILURE;

        // Copy the ear or war file to the WAS installableApps directory
        setSource(ivHelper.getUnpackedDir(this) + getAppFile());
        setTarget(getInstallableAppsDir());

        if (ivHelper.fileCopy(this)) {
            setMessage(getResourceString(TradeNLSKeys.COPYFILE_SUCCESS, getSource() + "," +
            getTarget()));
            rc = SUCCESS;
        } else {
            setMessage(getResourceString(TradeNLSKeys.COPYFILE_FAIL, getSource() + "," +
            getTarget()));
        }
    }

```

```

        rc = FAILURE;
    }

    ivHelper.log(this);
    return rc;
}

/**
 * install the Trade documents
 */
private void installTradeFiles()
{
    // note that copyDirectory does not return success or fail, assume success
    setSource(ivHelper.getUnpackedDir(this) + TradeLnxCommon.Trade_FILES);
    setTarget(getDocsDir());
    ivHelper.copyDirectory(this);
}

/**
 * Uses wsadmin to run the WebSphere jac1 scripts
 */
private int runWebSphereScript()
{
    int rc = SUCCESS;

    // Get the WAS script to be run
    // The script name is specified in the application properties file -
    DOCMGMT.WASscript
    String script = getWasScript();
    if ((script != null) && !script.trim().equals("")) {
        // The directory where the scripts are located are in the
        // unpackedDir. A subdirectory may have been specified for these in the
        // application.xml. The application properties file has this in
        DOCMGMT.scriptsDir
        String fullScriptsDir = ivHelper.getUnpackedDir(this) + getScriptsDir() + "/";

        // Run the WebSphere Admin application (wsadmin)
        // The -f option indicates a jac1 script is the next arg
        // After the script name, the args for the jac1 script are:
        // - directory where the scripts are located
        // - the fully qualified properties file name
        String [] command = {"/bin/sh", "-c",
            getWasProfileBinDir() +
                "wsadmin.sh -f " +
                fullScriptsDir + script + // Main jac1 script
                " " + fullScriptsDir + // the scripts directory
                " " + getResponseFileName()}; // and the properties file

        rc = invokeCommand(command);
        // Search the command result for error messages
        // If an error string is found then set rc=FAILURE
        String cmdResult = getCommandResult();
        if
        (TradeLnxCommon.messageExists(getCommandResult(), TradeLnxCommon.WAS_WSADMIN_ERRORS)) {
            rc = FAILURE;
        } else {
            // *****
            // *****
            // Now get the current module to server mapping
            // Run the WebSphere Admin application (wsadmin), the

```

```

        // -f option indicates a jacl script is the next arg
        // After the script name, the arg for the jacl script is:
        // - the application name
        script = TradeLnxCommon.GET_VIEW_SCRIPT;
        String [] command2 = {"/bin/sh", "-c",
            getWasProfileBinDir() +
                "wsadmin.sh -f " +
                    fullScriptsDir + script +      // Script to run
                    " " + getAppName()};           // the application name
        rc = invokeCommand(command2);

        // put the web server name together in the proper format
        // "WebSphere:cell=cellname,node=nodename,server=servername"
        // WebSphere:cell=$baseCell,node=$webNode,server=$webServerName
        // the webNode is in the format webServerName_node
        // get the cell name from the output of the view script
        int start = getCommandResult().indexOf("WebSphere:cell=");
        int end = getCommandResult().indexOf(",",start);
        String baseCell = getCommandResult().substring(start,end+1);
        // WebSphere:cell=vpriceNode01Cell
        String qualWebServerName = baseCell +
            "node=" + getIhsServerName() + "_node," +
            "server=" + getIhsServerName();

        // Ensure the view output has the correct keywords
        String updatedViewOutput =
        TradeLnxCommon.getModuleMapList(getCommandResult());
        if (updatedViewOutput != null) {
            // Now map the application modules to the web server
            // Run the WebSphere Admin application (wsadmin), the
            // -f option indicates a jacl script is the next arg
            // After the script name, the arg for the jacl script is:
            // - the application name
            // - the new module map list
            script = TradeLnxCommon.EDIT_MAP_SCRIPT;
            String [] command3 = {"/bin/sh", "-c",
                getWasProfileBinDir() +
                    "wsadmin.sh -f " +
                        fullScriptsDir + script +      // Script to run
                        " \" " + getAppName() + "\" " + // application name
                        " \" " + qualWebServerName + "\" " + // new web server name
                (formatted)
                    " \" " + updatedViewOutput + "\""};           //
            the application name
            rc = invokeCommand(command3);
        }

        // *****
        // *****
        }
    } else {
        setMessage(getResourceString(TradeNLSKeys.NO_WAS_SCRIPT));
        ivHelper.log(this);
    }

    return rc;
}

/**
 * Runs the DB2 Script

```



```

        */
private int runDB2Script()
{
    String cmdResult;

    int rc = SUCCESS;

    // Get the DB2 script to be run
    // The script name is specified in the application properties file -
    DOCMGMT.DB2script
    String script = getDB2Script();

    if ((script != null) && !script.trim().equals("")) {
        // The directory where the scripts are located are in the
        // unpackedDir. A subdirectory may have been specified for these in the
        // application.xml. The application properties file has this in
    DOCMGMT.scriptsDir
        String fullScriptsDir = ivHelper.getUnpackedDir(this) + getScriptsDir() + "/";

        // Check if the script is a shell ("sh") file
        String lowerCaseScript = script.toLowerCase();
        if (lowerCaseScript.endsWith(".sh")) {
            // Run shell cmd file
            // su command runs a shell with substitute user and group IDs
            // su [OPTION]...[-] [USER [ARG]...]
            // first argument passed will be the DB2 user id,
            // -c is to execute a single command

            // Run script passing in parameters

        /*
            String [] command1 = {"\" + fullScriptsDir + script + // Script path
                                " " + getDbName() + // database name
                                " " + fullScriptsDir + // Scripts
                                " " + getDB2InstanceId() + //
                                " " + getDB2PW() + "\"}";

            // document directory
        */

        // String [] command1 = { "/bin/su", "- ", "db2inst", "-c ", "echo xyz >
        /tmp/test" }; // Script path

        // String [] command1 = { "/bin/sh", "-c", "su - " + getDB2InstanceId() + " -c
        \" + "\" + fullScriptsDir + script + // Script path
        // " " + getDbName() + // database name
        // " " + fullScriptsDir + // Scripts directory
        // " " + getDB2InstanceId() + // DB2 userid
        // " " + getDB2PW() + "\" + "\"}";

        // String [] command1 = { "/bin/su", "-",
        "db2inst", "-c", "/irul/Trade_ScriptsDir/DB2Script.sh tradedb /irul/Trade_ScriptsDir/ +
        getDB2InstanceId() + getDB2PW() > /home/db2inst/DEBUG_OUTPUT"};
        String [] command1 = { "/bin/su", "-",
        "db2inst", "-c", "/irul/Trade_ScriptsDir/DB2Script.sh tradedb /irul/Trade_ScriptsDir/ +
        getDB2InstanceId() + " " + getDB2PW() + " > /home/db2inst/DEBUG_OUTPUT"};
        rc = invokeCommand(command1);
        cmdResult = getCommandResult();
        setMessage("Command result from db2script = " + cmdResult);
    }
}

```

```

        ivHelper.log(this);
    } else {
        // Assume this is an SQL file
        String [] command2 = {"/bin/sh", "-c", "su - " + getDB2InstanceId(), "-c \"db2
-f " + fullScriptsDir + script + "\""};
        rc = invokeCommand(command2);
        ivHelper.logAppendFile(this);
    }
} else {
    setMessage(getResourceString(TradeNLSKeys.NO_DB2_SCRIPT));
    ivHelper.log(this);
}
return rc;
}

/**
 * Invoke the command and log it
 * parm#0 = String [] command
 * This will set the command result in the instance
 * variable ivCommandResult in case the calling routine
 * needs to analyze the result
 *
 * The command and the result will both be logged here
 */
private int invokeCommand(String [] command)
{
    int rc = SUCCESS;

    try {
        setCommandResult(TradeLnXCommon.invokeCommand (this, ivHelper, command));
    } catch (Exception e) {
        rc = FAILURE;
        setMessage(getResourceString(TradeNLSKeys.CMD_EXCEPTION, e.toString()));
        ivHelper.log(this);
    }
    return rc;
}

/**
 * Get the install and bin directory for WAS
 * Set the installableApps directory based on the WASinstall directory
 */
private int determineWasDir()
{
    int rc = SUCCESS;

    setWasInstallDir(TradeLnXCommon.determineWasDir(this, ivHelper, getWasVpdUid()));
    if (getWasInstallDir() == null) {
        setMessage(getResourceString(TradeNLSKeys.WAS_NOT_INSTALLED));
        ivHelper.log(this);
        rc = FAILURE;
    } else {
        // Set the WAS profile bin dir
        setWasProfileBinDir(getWasInstallDir() +
            "profiles/" +
            getWasProfileName() + "/bin/");
        setInstallableAppsDir(getWasInstallDir() + "installableApps/"); // set the
installableApps directory
    }
}

```

```

        return rc;
    }

    /**
     * Get the install and bin directory for IHS
     */
    private int determineIHSDir()
    {
        int rc = SUCCESS;

        String ihsInstallDir = TradeLnxCommon.determineIhsDir(this, ivHelper,
getIhsVpdUid());
        if (ihsInstallDir == null) {
            setMessage(getResourceString(TradeNLSKeys.NO_HTTPSERVER));
            ivHelper.log(this);
            rc = FAILURE;
        } else {
            // Set the IHS bin dir
            setIHSBinDir(ihsInstallDir + "bin/");
        }

        return rc;
    }

    /**
     * Get the DB2 install directory
     */
    private int determineDB2Dir()
    {
        int rc = SUCCESS;
        setDB2InstallDir(TradeLnxCommon.determineDB2Dir(this, ivHelper, ivDB2Version));
        if (getDB2InstallDir() == null) {
            setMessage(getResourceString(TradeNLSKeys.DB2_NOT_INSTALLED));
            ivHelper.log(this);
            rc = FAILURE;
        }
        return rc;
    }

    /**
     * restart the HTTP Server
     */
    private int restartHTTPServer()
    {
        int rc = SUCCESS;

        //use the apachectl command to start the server, log an error but continue if an
error occurs
        String [] command2 = {"/bin/sh", "-c", getIHSBinDir() + "apachectl restart"};
        rc = invokeCommand(command2);

        if (rc != SUCCESS) {
            // log an error if not successful
            setMessage(getResourceString(TradeNLSKeys.START_HTTPSERVER_FAIL));
            ivHelper.log(this);
        }

        return rc;
    }
    /**

```

```

    * Generate the HTTP Plugin
    */
private int generateHTTPPlugin()
{
    int rc = SUCCESS;
    String [] command = {"/bin/sh", "-c", getWasProfileBinDir() + "GenPluginCfg.sh"};
    rc = invokeCommand(command);

    return rc;
}

/**
*****
*
* Standard getters and setters
*
*****
*/

/**
* Get the IHS Server name.
*/
private String getIhsServerName()
{
    return ivIhsServerName;
}

/**
* Set the IHS Server name.
*/
private void setIhsServerName(String serverName)
{
    ivIhsServerName = serverName;
}

/**
* Get the application name.
*/
private String getAppName()
{
    return ivAppName;
}

/**
* Set the application name.
*/
private void setAppName(String appName)
{
    ivAppName = appName;
}

/**
* Get the name of the ear or war file to be installed.
*/
private String getAppFile()
{
    return ivAppFile;
}

/**

```

```

    * Set the name of the ear or war file to be installed.
    */
private void setAppFile(String appName)
{
    ivAppFile = appName;
}

/**
 * Get the command result.
 */
private String getCommandResult()
{
    return ivCommandResult;
}

/**
 * Set the command result.
 */
private void setCommandResult(String commandResult)
{
    ivCommandResult = commandResult;
}

/**
 * Get the Database name
 */
private String getDbName()
{
    return ivDbName;
}

/**
 * Set the Database name
 */
private void setDbName(String dbName)
{
    ivDbName = dbName;
}

/**
 * Get the DB2 instance id.
 */
private String getDB2InstanceId()
{
    return ivDB2InstanceId;
}

/**
 * Set the DB2 instance id to use.
 */
private void setDB2InstanceId(String id)
{
    ivDB2InstanceId = id;
}

/**
 * Get the DB2 password.
 */
private String getDB2PW()
{
    return ivDB2PassWord;
}

```

```

}

/**
 * Set the DB2 password to use.
 */
private void setDB2PW(String pwd)
{
    ivDB2PassWord = pwd;
}

/**
 * Get the name of the DB2 script to be run.
 */
private String getDB2Script()
{
    return ivDB2Script;
}

/**
 * Set the name of the DB2 script to be run.
 */
private void setDB2Script(String fileName)
{
    ivDB2Script = fileName;
}

/**
 * Get the name of the WebSphere script to be run.
 */
private String getWasScript()
{
    return ivWasScript;
}

/**
 * Set the name of the WebSphere script to be run.
 */
private void setWasScript(String scriptName)
{
    ivWasScript = scriptName;
}

/**
 * Get the directory where the application documents are to be installed.
 */
private String getDocsDir()
{
    return ivDocsDir;
}

/**
 * Set the directory where the application documents are to be installed.
 */
private void setDocsDir(String docsDir)
{
    ivDocsDir = docsDir;
}

/**
 * Get the directory within userPrograms where the scripts are located
 */

```

```

private String getScriptsDir()
{
    return ivScriptsDir;
}

/**
 * Set the directory where the application documents are to be installed.
 */
private void setScriptsDir(String scriptsDir)
{
    ivScriptsDir = scriptsDir;
}

/**
 * Get the server name.
 */
private String getServerName()
{
    return ivWasServerName;
}

/**
 * Set the server name.
 */
private void setServerName(String serverName)
{
    ivWasServerName = serverName;
}

/**
 * Get the WAS VPD.properties UID.
 */
private String getWasVpdUid()
{
    return ivWasVpdUid;
}

/**
 * Set the WAS vpd UID
 */
private void setWasVpdUid(String uid)
{
    ivWasVpdUid = uid;
}

/**
 * Get the IHS VPD.properties UID.
 */
private String getIhsVpdUid()
{
    return ivIhsVpdUid;
}

/**
 * Set the WAS vpd UID
 */
private void setIhsVpdUid(String uid)
{
    ivIhsVpdUid = uid;
}

```

```

    }

    /**
     * Get the WAS profile name.
     */
    private String getWasProfileName()
    {
        return ivWasProfileName;
    }

    /**
     * Set the WAS profile name.
     */
    private void setWasProfileName(String profName)
    {
        ivWasProfileName = profName;
    }

    /**
     * Get the WAS bin directory name.
     */
    private String getWasProfileBinDir()
    {
        return ivWasProfileBinDir;
    }

    /**
     * Set the WAS bin directory name.
     */
    private void setWasProfileBinDir(String binDir)
    {
        ivWasProfileBinDir = binDir;
    }

    /**
     * Get the WAS install directory name.
     */
    private String getWasInstallDir()
    {
        return ivWasInstallDir;
    }

    /**
     * Set the WAS install directory name.
     */
    private void setWasInstallDir(String installDir)
    {
        ivWasInstallDir = installDir;
    }

    /**
     * Get the WAS installableApps directory name.
     */
    private String getInstallableAppsDir()
    {
        return ivInstallableAppsDir;
    }

    /**

```



```

    * Set the WAS installableApps directory name.
    */
private void setInstallableAppsDir(String installableAppsDir)
{
    ivInstallableAppsDir = installableAppsDir;
}

/**
 * Get a DB2 version to be search in the rpm database.
 */
private String getDB2Version()
{
    return ivDB2Version;
}

/**
 * Set the WAS vpd UID
 */
private void setDB2Version(String ver)
{
    ivDB2Version = ver;
}

/**
 * Get the DB2 install directory name.
 */
private String getDB2InstallDir()
{
    return ivDB2InstallDir;
}

/**
 * Set the DB2 install directory name.
 */
private void setDB2InstallDir(String installDir)
{
    ivDB2InstallDir = installDir;
}

/**
 * Set the IHS bin directory.
 */
private void setIHSBinDir(String binDir)
{
    ivIHSBinDir = binDir;
}

/**
 * Get the IHS bin directory.
 */
private String getIHSBinDir()
{
    return ivIHSBinDir;
}

private String getDB2PassWord() {
    return ivDB2PassWord;
}

private void setDB2PassWord(String ivDB2PassWord) {
    this.ivDB2PassWord = ivDB2PassWord;
}

```

```

        private String getDB2UserId() {
            return ivDB2UserId;
        }
        private void setDB2UserId(String ivDB2UserId) {
            this.ivDB2UserId = ivDB2UserId;
        }
    }

    /**
     * ***** Main Routine *****
     */
    public static void main(String[] args)
    {
        TradeLnxMain TradeMain = new TradeLnxMain();

        int retVal = TradeMain.init(args);
        if (retVal == SUCCESS)
            retVal = TradeMain.install();

        System.exit(retVal);
    }
}

```

The TradeLnxPDC.java program

Example B-4 is a sample TradeLnxPDC.java file which was used for deploying the Trade6 application.

Example: B-4 TradeLnxPDC.java

```

package com.trade;
/**
 * @(#)SampleLnxPDC.java
 *
 * Licensed Materials - Property of IBM
 *
 * 5724-J10
 *
 * (C) Copyright IBM Corp. 2005
 *
 * US Government Users Restricted Rights - Use, duplication or disclosure
 * restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * This is sample code made available for use in accordance with terms set
 * forth in the license agreement document for the IBM Express Runtime.
 */

import com.trade.TradeNLSKeys;
import com.ibm.jsdt.support.SupportLinuxBase;
import com.ibm.jsdt.support.SupportLinuxHelper;

public class TradeLnxPDC extends SupportLinuxBase
{
    private static final String copyright0="Licensed Materials - Property of IBM";
    private static final String copyright1="5724-J10";
    private static final String copyright2="(C) Copyright IBM Corp. 2005 All Rights
    Reserved.";
}

```

```
private static final String copyright3="US Government Users Restricted Rights - Use,
duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.";
```

```
private String ivCommandResult = null;
private String ivWASInstallDir = null;
private String ivWASProfileBinDir = null;
private String ivAppName = null;
private String ivWASServerName = null;
private String ivWasProfileName = null;
private String ivWasVpdUid = null;
private String ivDb2Version = null;
private String ivScriptsDir = null;
private SupportLinuxHelper ivHelper = null;
```

```
/**
 * Initializes the support framework and checks that the right number of arguments were
 * passed.
```

```
 * @param args
 * @return SUCCESS if the right number of parameters were passed, otherwise FAILURE
 */
```

```
private int init (String args[])
```

```
{
    int rc = SUCCESS;
    ivHelper = getLinuxHelper();
    setMainResources(TradeLnCommon.Trade_MESSAGES);
```

```
    setMessage("beginning of PDC");
```

```
    ivHelper.postWarningMessageToDeployerUI(this);
```

```
    if (args.length != 1)
```

```
{
        setMessage(getResourceString(TradeNLSKeys.BAD_NUMBER_PGM_ARGS, new String[] {
            "TradeLnXPDC",
            "1",
            Integer.toString(args.length) }));
        ivHelper.log(this);
        rc = FAILURE;
        setMessage("beginning of 1");
        ivHelper.postWarningMessageToDeployerUI(this);
    }
```

```
    else
```

```
{
        // The properties file is in the same format as a response file
        setResponseFileName(args[0]);
        setMessage("beginning of 2");
        ivHelper.postWarningMessageToDeployerUI(this);
    }
```

```
    return rc;
}
```

```
/**
```

```
 * The Trade application requires WAS and DB2. This PDC will
 * - check to see if WAS is installed (need WAS vpd.properties unique identifier (UID)
 * specified in the properties file).
```

```
 * - check to see if DB2 is installed (need DB2 version as it is found in the rpm
 * database specified in the properties file).
```

```
 * - If both are installed, it will determine if the application is
 * already installed. It will use a jac1 script via wsadmin, the results being put
```

```

*   into a string - ivCommandResult which will be searched for errors.
*   - if there is a WAS error returned, it could be that WAS is not running, therefore
*     the WAS start command will be run and the check to determine if the app
*     is already installed will be run again.
*   - There are four possible results in the command result string:
*     - APP_EXISTS indicating the application is already installed - return PDC_EXISTS
*     - APP_DOES_NOT_EXIST - return PDC_DOES_NOT_EXISTS
*     - WASXXXX messages indicating the WAS server is not started - return FAILURE
*     - the result file does not exist or some other, unexpected error - return FAILURE
*/
private int check()
{
    int rc = SUCCESS;
    int resultRc = FAILURE; // init to return a failure result
    boolean startedWAS = false;

    // get the values set in the properties file, properties file is mandatory
    if (getResponseFileName() == null || getResponseFileName().trim().equals("")) {
        setMessage(getResourceString(TradeNLSKeys.NO_PROPERTIES_FILE));
        ivHelper.log(this);
        rc = FAILURE;
        setMessage("End of 3");
        ivHelper.postWarningMessageToDeployerUI(this);
    } else {
        rc = getProperties();
        setMessage("End of 1");
        ivHelper.postWarningMessageToDeployerUI(this);
    }

    // Determine if WAS is installed and get the WAS directories
    if (rc == SUCCESS) {
        rc = determineWasDir();
        setMessage("End of 1");
        ivHelper.postWarningMessageToDeployerUI(this);
    }

    // Determine if DB2 is installed
    if (rc == SUCCESS) {
        String db2InstallDir = null;
        db2InstallDir = TradeLnxCommon.determineDB2Dir(this, ivHelper, ivDb2Version);
        if (db2InstallDir == null || db2InstallDir.trim().equals("")) {
            setMessage(getResourceString(TradeNLSKeys.DB2_NOT_INSTALLED));
            ivHelper.log(this);
            setMessage("End of 2");
            ivHelper.postWarningMessageToDeployerUI(this);
            rc = FAILURE;
        }
    }

    if (rc == SUCCESS) {

        // Determine if the application is already installed
        // Run the WebSphere Admin application (wsadmin), the
        // -f option indicates a jac1 script is the next arg
        // After the script name, the arg for the jac1 script is:
        // - the application name

        String unpackedDir = ivHelper.getUnpackedDir(this);
    }
}

```

```

String [] checkCommand = {"/bin/sh", "-c", ivWASProfileBinDir + "wsadmin.sh -f "
    + unpackedDir + getScriptsDir()
    + TradeLnxCommon.CHECK_INSTALL_SCRIPT //script to run
    + " " + getAppName()}; //the application name

// invoke and log the command
rc = invokeCommand(checkCommand);

if (rc == SUCCESS) {
    // set up directory from which to issues WAS command
    // Search the command result for unique strings indicating a wsadmin error
    // If any of these errors occur, it may be that WAS is not started
    // Start the server using the command in the bin directory of the specified
profile.
    if
(TradeLnxCommon.messageExists(getCommandResult(),TradeLnxCommon.WAS_WSADMIN_ERRORS)) {
        String [] startCommand = {"/bin/sh", "-c", ivWASProfileBinDir +
"startServer " + getServerName()};
        setMessage("End of 3");
        ivHelper.postWarningMessageToDeployerUI(this);

        rc = invokeCommand(startCommand);

        // Check the result of the start command
        if
(TradeLnxCommon.messageExists(getCommandResult(),TradeLnxCommon.WAS_START_ERRORS)) {
            // error starting the server
            rc = FAILURE;
        } else {
            startedWAS = true;
            // and run the check install script again.
            rc = invokeCommand(checkCommand);

            // check for wsadmin errors
            if
(TradeLnxCommon.messageExists(getCommandResult(),TradeLnxCommon.WAS_WSADMIN_ERRORS)) {
                ivHelper.postWarningMessageToDeployerUI(this);
                rc = FAILURE; // set a FAILURE return code if errors
            }
        }
    }

    // Get the result from the jacl script
    if (rc == SUCCESS) {
        if(getCommandResult().indexOf(TradeLnxCommon.APP_EXISTS) >= 0) {
            setMessage("End of 4");
            ivHelper.postWarningMessageToDeployerUI(this);
            resultRc = PDC_EXISTS;
        }
        if (startedWAS == true) {
            // put WAS back to its original state
            String [] stopCommand = {"/bin/sh", "-c", ivWASProfileBinDir +
"stopServer " + getServerName()};
            invokeCommand(stopCommand);
        }
        if(getCommandResult().indexOf(TradeLnxCommon.APP_DOES_NOT_EXIST) >= 0) {
            resultRc = PDC_DOES_NOT_EXIST;
        }
    }
}

```

```

    }
    setMessage("End of PDC");
    ivHelper.postWarningMessageToDeployerUI(this);

    return resultRc;
}
/**
 * Get values from the properties file
 */
private int getProperties()
{
    int rc = SUCCESS;
    // The properties file name was set in main

    // Log the contents of the properties file
    ivHelper.logNewLine(this);
    setFileName(getResponseFileName());
    ivHelper.logAppendFile(this);
    ivHelper.logNewLine(this);

    // get the values specified in the properties file that are needed by this PDC

    //set the WAS UID used in the ISMP vpd.properties file, needed to get the install
    directory
    setKey(TradeLnxCommon.WAS_VPD_UID);
    setWasVpdUid(ivHelper.getResponseFileValue(this));
    if (getWasVpdUid() == null || getWasVpdUid().trim().equals("")) {
        setMessage(getResourceString(TradeNLSKeys.NO_PROPERTY, TradeLnxCommon.WAS_VPD_UID));
        ivHelper.log(this);
        rc = FAILURE;
    }

    setKey(TradeLnxCommon.SERVER_NAME_KEY); // server name to use for the start command
    setServerName(ivHelper.getResponseFileValue(this));
    if (getServerName() == null || getServerName().trim().equals("")) {
        setMessage(getResourceString(TradeNLSKeys.NO_PROPERTY,
TradeLnxCommon.SERVER_NAME_KEY));
        ivHelper.log(this);
        rc = FAILURE;
    }

    setKey(TradeLnxCommon.WAS_PROFILE_KEY); // WAS profile name
    setWasProfileName(ivHelper.getResponseFileValue(this));
    if (ivWasProfileName == null || ivWasProfileName.trim().equals("")) {
        setMessage(getResourceString(TradeNLSKeys.NO_PROPERTY,
TradeLnxCommon.WAS_PROFILE_KEY));
        ivHelper.log(this);
        rc = FAILURE;
    }

    //set the DB2 version as it appears in the rpm database, needed to get the install
    directory
    setKey(TradeLnxCommon.DB2_VERSION);
    setDb2Version(ivHelper.getResponseFileValue(this));
    if (getDb2Version() == null || getDb2Version().trim().equals("")) {
        setMessage(getResourceString(TradeNLSKeys.NO_PROPERTY, TradeLnxCommon.DB2_VERSION));
        ivHelper.log(this);
        rc = FAILURE;
    }
}

```

```

        setKey(TradeLnxCommon.APP_NAME_KEY);// the name of the application to be installed, it
is required
        setAppName(ivHelper.getResponseFileValue(this));
        if (getAppName() == null || getAppName().trim().equals("")) {
            setMessage(getResourceString(TradeNLSKeys.NO_PROPERTY,
TradeLnxCommon.APP_NAME_KEY));
            ivHelper.log(this);
            rc = FAILURE;
        }

        setKey(TradeLnxCommon.SCRIPTS_DIR_KEY);// the name of the scripts directory
        setScriptsDir(ivHelper.getResponseFileValue(this));
        if (getScriptsDir() == null || getScriptsDir().trim().equals("")) {
            setScriptsDir(" ");
        } else
            // Add a slash to the end of the path if one doesn't exist
            if (!getScriptsDir().endsWith("/"))
            {
                setScriptsDir(getScriptsDir() + "/");
            }

return rc;

}

/**
 * Invoke the command, the command will also be logged
 *
 * This will set the command result in an instance
 * variable in case the calling routine needs to
 * analyze the result
 */
private int invokeCommand(String [] command)
{
    int rc = SUCCESS;

    try {
        setCommandResult(TradeLnxCommon.invokeCommand (this, ivHelper, command));
    } catch (Exception e) {
        rc = FAILURE;
        setMessage(getResourceString(TradeNLSKeys.CMD_EXCEPTION, e.toString()));
        ivHelper.log(this);
    }
    return rc;
}

/**
 * Get the install and bin directory for WAS
 * Set the installableApps directory based on the WASinstall directory
 */
private int determineWasDir()
{
    int rc = SUCCESS;

    setWasInstallDir(TradeLnxCommon.determineWasDir(this, ivHelper, getWasVpdUid()));
    if (ivWASInstallDir == null) {
        setMessage(getResourceString(TradeNLSKeys.WAS_NOT_INSTALLED));
        ivHelper.log(this);
        rc = FAILURE;
    }
}

```

```

    } else {
        // Set the WAS profile bin dir
        setWasProfileBinDir(ivWASInstallDir +
            "profiles/" +
            getWasProfileName() + "/bin/");
    }

    return rc;
}

/**
 *
 * Getters and Setters
 */

/**
 * Get the command result.
 */
private String getCommandResult()
{
    return ivCommandResult;
}

/**
 * Set the command result.
 */
private void setCommandResult(String commandResult)
{
    ivCommandResult = commandResult;
}

/**
 * Get the application name.
 */
private String getAppName()
{
    return ivAppName;
}

/**
 * Set the application name.
 */
private void setAppName(String appName)
{
    ivAppName = appName;
}

/**
 * Get the server name.
 */
private String getServerName()
{
    return ivWASServerName;
}

/**
 * Set the server name.
 */
private void setServerName(String serverName)
{

```



```

        ivWASServerName = serverName;
    }

    /**
     * Get the WAS profile name.
     */
    private String getWasProfileName()
    {
        return ivWasProfileName;
    }

    /**
     * Set the WAS profile name.
     */
    private void setWasProfileName(String profName)
    {
        ivWasProfileName = profName;
    }

    /**
     * Get the WAS VPD.properties UID.
     */
    private String getWasVpdUid()
    {
        return ivWasVpdUid;
    }

    /**
     * Set the WAS vpd UID
     */
    private void setWasVpdUid(String uid)
    {
        ivWasVpdUid = uid;
    }

    /**
     * Get the WAS profile bin directory name.
     */
    private String getWasProfileBinDir()
    {
        return ivWASProfileBinDir;
    }

    /**
     * Set the WAS profile bin directory name.
     */
    private void setWasProfileBinDir(String binDir)
    {
        ivWASProfileBinDir = binDir;
    }

    /**
     * Get the WAS install directory name.
     */
    private String getWasInstallDir()
    {
        return ivWASInstallDir;
    }

    /**

```

```

    * Set the WAS install directory name.
    */
private void setWasInstallDir(String installDir)
{
    ivWASInstallDir = installDir;
}

/**
 * Get the WAS VPD.properties UID.
 */
private String getDb2Version()
{
    return ivDb2Version;
}

/**
 * Set the WAS vpd UID
 */
private void setDb2Version(String ver)
{
    ivDb2Version = ver;
}

/**
 * Get the directory within userPrograms where the scripts are located
 */
private String getScriptsDir()
{
    return ivScriptsDir;
}

/**
 * Set the directory where the application documents are to be installed.
 */
private void setScriptsDir(String scriptsDir)
{
    ivScriptsDir = scriptsDir;
}

/**
 * ***** Main Routine *****
 * @param args
 * 1 - properties file
 */
public static void main(String args[])
{
    TradeLnxPDC checker = new TradeLnxPDC();

    int retVal = checker.init(args);
    if (retVal == SUCCESS)
        retVal = checker.check();

    System.exit(retVal);
}
}

```

The TradeLnxCommon.java program

Example B-5 is a sample TradeLnxCommon.java file which was used for deploying the Trade6 application.

Example: B-5 TradeLnxCommon.java

```
package com.trade;
/**
 * @(#) SampleLnxCommon.java
 *
 * Licensed Materials - Property of IBM
 *
 * 5724-J10
 *
 * (C) Copyright IBM Corp. 2005
 *
 * US Government Users Restricted Rights - Use, duplication or disclosure
 * restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * This is sample code made available for use in accordance with terms set
 * forth in the license agreement document for the IBM Express Runtime.
 */

import com.trade.TradeNLSKeys;
import com.ibm.jsdt.support.SupportLinuxBase;
import com.ibm.jsdt.support.SupportLinuxHelper;

/**
 * This class contains constants and methods referenced by all user programs in the Sample
 * Wrapper.
 */
public class TradeLnxCommon {
    public static final String copyright0="Licensed Materials - Property of IBM";
    public static final String copyright1="5724-J10";
    public static final String copyright2="(C) Copyright IBM Corp. 2005 All Rights
Reserved.";
    public static final String copyright3="US Government Users Restricted Rights - Use,
duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.";

    public static final String WASEXPRESS_NAME = "WebSphere Application Server, Express";
    public static final String Trade_MESSAGES = "com.trade.TradeMessagesNLS";
    public static final String Trade_FILES = "RuntimeDocs";
    public static final String CHECK_INSTALL_SCRIPT = "CheckAppInstall.jacl";
    public static final String APP_EXISTS = "APP_EXISTS"; // used in the checkInstall
script
    public static final String APP_DOES_NOT_EXIST = "APP_DOES_NOT_EXIST"; // used in the
checkInstall script
    public static final String[] WAS_WADMIN_ERRORS = {
        "WASX7023E", // Error creating connection to host
        "WASX7213I", // The scripting client is not connected to a server process
        "WASX7309W", // No "save" was performed before the script exited
        "WASX7017E"}; // Exception received while running script
    public static final String[] WAS_START_ERRORS = {
        "ADMU0111E"}; // Program exiting with error

    // application response file keys
    public static final String WAS_VPD_UID = "WAS.vpdUID";
    public static final String IHS_VPD_UID = "IHS.vpdUID";
    public static final String SERVER_NAME_KEY = "WAS.serverName";
```

```

public static final String IHS_SERVER_NAME_KEY = "IHS.webServerName";
public static final String WAS_SCRIPT_KEY = "TRADE.WASscript";
public static final String GET_VIEW_SCRIPT = "IRU_ViewApp.jacl";
public static final String EDIT_MAP_SCRIPT = "IRU_EditMap.jacl";
public static final String APP_NAME_KEY = "WAS.appName";
public static final String WAS_PROFILE_KEY = "WAS.profile";
public static final String DB2_SCRIPT_KEY = "TRADE.DB2script";
public static final String DOCS_DIR_KEY = "DOCMGMT.documentsDir";
public static final String SCRIPTS_DIR_KEY = "TRADE.scriptsDir";
public static final String DB2_VERSION = "DB2.version";
public static final String DB_NAME_KEY = "DB2.databaseName";
public static final String DB2_INSTANCE_ID_KEY = "DB2.UserID";
public static final String DB2_INSTANCE_PW_KEY = "DB2.UserPW";
public static final String JDBC_CLASSPATH_KEY = "WAS.JDBCclasspath";
public static final String WAS_INSTALLABLE_APPS_DIR_KEY = "WAS.installableAppsDir";
public static final String APP_FILE_KEY = "WAS.appInstallFile";
public static final String MODULELABEL = "Module:";
public static final String URILABEL = "URI:";
public static final String SERVERLABEL = "Server:";

/**
 * Invoke the command, and log it
 * Both the command and the command result will be logged here
 *
 * @param base, SupportLinuxBase
 * @param helper, SupportLinuxHelper
 * @param command
 * @return the command result
 */
public static String invokeCommand(SupportLinuxBase base, SupportLinuxHelper helper,
String [] command) throws Exception
{
    String commandResult = null;

    base.setCommandArray(command);
    //log the command
    base.setMessage(base.getResourceString(TradeNLSKeys.CMDINVOKED,
helper.getStringFromCommandArray(base)));
    helper.log(base);
    helper.logNewLine(base);

    // invoke the command and set the command result to be returned
    // this may throw an exception
    commandResult = helper.getSystemCommandOutput(base);
    base.setMessage(commandResult);
    helper.log(base);

    return commandResult;
}

/**
 * Get the install directory for WAS
 * The returned directory will end with a slash
 * @param base, SupportLinuxBase
 * @param helper, SupportLinuxHelper
 * @param wasVpdUid (WAS vpd.properties unique identifier)
 * @return the WAS install directory
 */

```

```

    public static String determineWasDir(SupportLinuxBase base, SupportLinuxHelper helper,
String wasVpdUid)
    {
        String wasInstallDir = null;
        if (wasVpdUid !=null && !wasVpdUid.equals(""))
        {
            base.setVpdUid(wasVpdUid);
            String [] wasLocations = helper.getVpdInstallLocationArray(base);
            //this returns the location of the last WAS installed, in case more than one WAS
is installed
            if(wasLocations != null)
            {
                wasInstallDir = wasLocations[0];
                //Add a slash to the end of the path if one doesn't exist
                if (!wasInstallDir.endsWith("/")) {
                    wasInstallDir = wasInstallDir + "/";
                }
            }
        }

        return wasInstallDir;
    }

/**
 * Get the install directory for IHS
 * The returned directory will end with a slash
 * @param base, SupportLinuxBase
 * @param helper, SupportLinuxHelper
 * @param ihsVpdUid, String (IHS vpd.properties unique identifier)
 * @return the IHS install directory
 */
    public static String determineIhsDir(SupportLinuxBase base, SupportLinuxHelper helper,
String ihsVpdUid)
    {
        String ihsInstallDir = null;
        if (ihsVpdUid !=null && !ihsVpdUid.equals(""))
        {
            base.setVpdUid(ihsVpdUid);
            String [] ihsLocations = helper.getVpdInstallLocationArray(base);
            //this returns the location of the last IHS installed, in case more than one IHS
is installed
            if(ihsLocations != null)
            {
                ihsInstallDir = ihsLocations[0];
                if (!ihsInstallDir.endsWith("/")) {
                    ihsInstallDir = ihsInstallDir + "/";
                }
            }
        }

        return ihsInstallDir;
    }

/**
 * Get the install directory for DB2
 * The returned directory will end with a slash
 * @param base, SupportWindowsBase
 * @param helper, SupportWindowsHelper
 * @param db2Version, String (as it appears in the rpm database)
 * @return the DB2 install directory
 */

```

```

    public static String determineDB2Dir(SupportLinuxBase base, SupportLinuxHelper helper,
String db2Version)
    {
        String db2InstallDir = null;
        base.setRpmPackage(db2Version);
        if (helper.isRpmInstalled(base)) {
            db2InstallDir = "/opt/IBM/db2/V8.1/";
        }

        return db2InstallDir;
    }
/**
 * Determine if one or more messages exist in a string
 * @param source, the string to check
 * @param msgs, an array of message numbers
 * @return boolean, true if a message is found in the string
 */
public static boolean messageExists(String source,String [] msgs) {
    for (int i = 0; i < msgs.length; i++) {
        if (source.indexOf(msgs[i]) >= 0) {
            return true;
        }
    }
    return false;
}
/**
 * Helper method to parse the source string looking for sets of MODULE, URI, and SERVER
values.
 * @param source, the string to check
 * @param server, the server name to add if not already in the server list
 * @return String, A Space delimited string of module uri server values or null if none
 * found.
 */
public static String getModuleMapList(String source)
{
    //This routine assumes the source string contains data returned from issuing the
wsadmin
    //command of $AdminApp view DefaultApplication "-MapModulesToServers" . Further,
the
    //contents of the source string is assumed to be in a specific order in the
following format:
    //Module: value of module\n
    //URI: value of URI\n
    //Server: value of Server\n
    //Because of issues found with the data returned from this call the only assumption
we can make
    //is the label of URI: is NOT translated. We must assume the other labels for
Module: and Server:
    //are translated and thus can not be used in the search algorithm. The search
algorithm utilized
    //in this method is not the most elegant but it did provide a workaround for the
translated label
    //issue. The output list is space delimited where module value is enclosed in {}
and both URI and
    //server values contain no spaces. If the current server value from source does not
contain the
    //server value passed into the method, append the server value to the server list with
a + sign and
    //no spaces in the value.

```

```

//EX.
WebSphere:cell=myNode01Cell,node=myNode01,server=server1+WebSphere:cell=myNode01Cell,node=webserver1_node,server=webserver1
String value = "";
int moduleStart = 0, moduleEnd = 0, serverStart = 0, serverEnd = 0, totalLength = 0,
previousLineLength = 0, sourcePointer = 0;
String[] lines = source.split("\n");

for (int i = 0; i < lines.length; i++)
{
    if (-1 != lines[i].indexOf(URILABEL))
    {
        if ((0 != i) && (i + 1 < lines.length))
        {
            moduleStart = totalLength;
            moduleEnd = moduleStart + lines[i-1].indexOf(":") + 1;
            serverStart = totalLength + lines[i-1].length() + 1 + lines[i].length()
+ 1;

            serverEnd = serverStart + lines[i+1].indexOf(":") + 1;
            if(0 != sourcePointer)
            {
                value += source.substring(sourcePointer, moduleStart);
            }
            value += MODULELABEL;
            value += source.substring(moduleEnd, serverStart);
            value += SERVERLABEL;
            sourcePointer = serverEnd;
        }
    }
    totalLength += previousLineLength;
    previousLineLength = lines[i].length() + 1; // Add one for new line character
}
value += source.substring(sourcePointer, source.length());
return (value);
}

}

```

The CheckAppInstall.jacl script file

Example B-6 is a sample CheckAppInstall.java file which was used for deploying the Trade6 application.

Example: B-6 CheckAppInstall.jacl script

```

# %I% %W% %G% %U%
# *****

# CheckAppInstall will list all installed apps and put out
# the text APP_EXISTS if the application name in the arg is found or
# the text APP_DOES_NOT_EXIST if the application name is not found
# One arg is expected, the name of the application

global AdminApp
global env
set appList [$AdminApp list]

```

```

set appArg [lindex $argv 0]
set result APP_DOES_NOT_EXIST

foreach appName $appList {
    puts $appName
    if {$appName == $appArg} {
        set result APP_EXISTS
        break
    }
}
puts $result

```

The WebsphereConfigProcs.jacl script file

Example B-7 is a sample WebSphereConfigProcs.jacl file which was used for deploying the Trade6 application.

Example: B-7 WebSphereConfigProcs.jacl file

```

# %I% %W% %G% %U%
# Licensed Materials - Property of IBM
#
# 5724-F71 5724-J10
#
# (C) Copyright IBM Corporation 2004, 2005 All Rights Reserved
#
# US Government Users Restricted Rights- Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM
# Corp.
#
# This is sample code made available for use in accordance with
# terms set forth in the license agreement document for the IBM
# Express Runtime.

#-----
# Config procs
#-----

# JACL NOTES
# Braces inside comments are still matched.
# What that means is that you must provide a matching end brace if you want to
# comment out a line e.g.
# if { condition } { ;# you must provide the ending brace even tho this line is commented
# }

#-----
#
# createdB2JDBCProvider - this takes base node, the server name,
#                        the JDBCProvider name, the classpath to the
#                        JDBCProvider code, the name of the implementation
#                        class, the xa setting, and a description
# Parameters:
#   bn - baseNode - typically DefaultNode
#   serv - server name - typically server1
#   provName - name of the JDBC provider - can be any string

```



```

# classpath - path to and including the db2java.zip file e.g. C:/sqllib/java/db2java.zip
# implClass - implementation class name of the JDBC driver - typically
COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource
# xa - whether this datasource is single or 2 phase - true/false
# desc - free form description string
#
# NOTES:
# - nativepath - not used in this proc
# - propertySet - not used in this proc
#
#-----
proc createDB2JDBCProvider {bn serv provName classPath implClass xa desc} {
    puts "\nConfigProcs: createDB2JDBCProvider $bn $serv provName classPath implClass xa
desc"
    global AdminConfig ;# Access the AdminConfig command

    if {[file exists $classPath]} {
        # get the path name upto the driver to use for DB2_JDBC_DRIVER_PATH
        set db2jdbcdriverpath [file dirname $classPath]
        # set the WebSphere environment variable DB2_JDBC_DRIVER_PATH at node scope
        updateVariableMap DB2_JDBC_DRIVER_PATH $db2jdbcdriverpath
        # use the WebSphere environment variable DB2_JDBC_DRIVER_PATH in the JDBC provider
        set jdbcProvClasspath [file join \$\{DB2_JDBC_DRIVER_PATH\} [file tail
$classPath]]
    } else {
        return -code error "Could not find the JDBC driver at the location provided
$classPath"
    }

    #-----
    # Get the config id of the server
    #-----
    set parent [$AdminConfig getid /Node:$bn/]

    #-----
    # Set the variables for the JDBCProvider, use path and name provided
    # in the arguments
    #-----
    set pname_attr [list name $provName]
    set path_attr [list classpath $jdbcProvClasspath]
    set impl_attr [list implementationClassName $implClass]
    set desc_attr [list description $desc]
    set xa_attr [list xa $xa]
    set jdbcAttrs [list $pname_attr $path_attr $impl_attr $xa_attr $desc_attr]
    puts "\nConfigProcs: create JDBCProvider $parent $jdbcAttrs"
    $AdminConfig create JDBCProvider $parent $jdbcAttrs
}

# createJDBCProviderUsingTemplate - creates a new DB2 type 2 legacy provider at the server
scope
#
# This method has the advantage of delegating most of the responsibility for
# knowing the right implementation class name and other driver details to
# WebSphere itself.
#
# Parameters
#         provName - name of the provider - any text string
#         classpath - path to and including the db2java.zip file e.g.
C:/sqllib/java/db2java.zip

```

```

#           serverName - optional - defaults to server1
#           nodeName - optional - defaults to DefaultNode
#
proc createJDBCProviderUsingTemplate {provName classPath nodeName serverName } {
    puts "\nConfigProcs: createJDBCProviderUsingTemplate $provName $classPath $nodeName
$serverName"
    global AdminConfig ;# Access the AdminConfig command

    if {[file exists $classPath]} {
        # get the path name upto the driver to use for DB2_JDBC_DRIVER_PATH
        set db2jdbcdriverpath [file dirname $classPath]
        # set the WebSphere environment variable DB2_JDBC_DRIVER_PATH at node scope
        # updateVariableMap DB2_JDBC_DRIVER_PATH $db2jdbcdriverpath $serverName $nodeName
        updateVariableMap DB2UNIVERSAL_JDBC_DRIVER_PATH $db2jdbcdriverpath $serverName
$nodeName
    } else {
        return -code error "Could not find the JDBC driver at the location provided
$classPath"
    }

    set parent [$AdminConfig getid /Node:$nodeName/]
    set pname_attr [list name $provName]
    set attrList [list $pname_attr]
    # set temp1 [$AdminConfig listTemplates JDBCProvider "DB2 Legacy CLI-based Type 2 JDBC
Driver("]
    set temp1 [$AdminConfig listTemplates JDBCProvider "DB2 Universal JDBC Driver Provider
("]
    # $AdminConfig createUsingTemplate JDBCProvider $parent {{name $provName}} $temp1
    puts " "
    puts "ConfigProcs: AdminConfig createUsingTemplate JDBCProvider $parent $attrList
$temp1"
    puts " "
    $AdminConfig createUsingTemplate JDBCProvider $parent $attrList $temp1

    # Create by template automatically creates a 4.0 and 5.0 DS with no name
    # We could modify them to suit us but it adds to the complexity of the main script
    # so just delete the extra template datasources
    # ignore any errors

    catch [set dsToDelete [$AdminConfig getid /JDBCProvider:$provName/DataSource:]]
    catch [$AdminConfig remove $dsToDelete]

    catch [set dsToDelete [$AdminConfig getid /JDBCProvider:$provName/WAS40DataSource:]]
    catch [$AdminConfig remove $dsToDelete]

    puts "Please remember to issue a \ $AdminConfig save if you wish to save this
configuration change"
}

#-----
# createJAASDataAuth - this is used for connection to the
#                   database and authorization for the database
#
# JAASDataAuth attributes are:
# "alias String"
# "description String"
# "password String"

```

```

# "userId String"
#-----
proc createJAASDataAuth {aliasName user pw desc} {
    global AdminConfig

    set security [$AdminConfig list Security]
    set jaasAlias_attr [list alias $aliasName]
    set jaasDesc_attr [list description $desc]
    set user_attr [list userId $user]
    set pw_attr [list password $pw]
    set jaas_attrs [list $jaasAlias_attr $user_attr $pw_attr $jaasDesc_attr]
    # create special attributes to be logged - we do not want to log the password
    set pw_attr_log [list password ****]
    set jaas_attrs_log [list $jaasAlias_attr $user_attr $pw_attr_log $jaasDesc_attr]
    puts " "
    puts "ConfigProcs: AdminConfig create JAASAuthData $security $jaas_attrs_log"
    puts " "
    $AdminConfig create JAASAuthData $security $jaas_attrs
}

#-----
#
# createdB2DataSource - this takes base node, the server name,
#                       the JDBCProvider name to which this DataSource
#                       is to be associated,
# NOTE: A ConnectionPool will also be created within this proc
#
# DataSource attributes are:
# "authDataAlias String"
# "authMechanismPreference ENUM(BASIC_PASSWORD, KERBEROS)"
# "category String"
# "connectionPool ConnectionPool"
# "datasourceHelperClassname String"
# "description String"
# "jndiName String"
# "mapping MappingModule"
# "name String"
# "provider J2EEResourceProvider@"
# "relationalResourceAdapter J2CResourceAdapter@"
# "statementCacheSize Integer"
# "propertySet J2EEResourcePropertySet"
#
# NOTES:
# - mapping is not used
# - we will set the JNDI name as "jdbc/" with the datasource name appended
# - the property set for a DB2 DataSource includes
#   database name, this is a required attribute
# - a relational resource adapter will be created using
#   the "WebSphere Relational Resource Adapter"
#-----
proc createDB2DataSource {bn serv JDBCProvName dsName dbName dsHelper authAlias authMech
cat desc} {
    global AdminConfig

    set jdbcProvId [$AdminConfig getid /JDBCProvider:$JDBCProvName/]

    #-----
    # Set up the properties for a DB2 DataSource

```

```

#-----
set dbname_attr [list [list name dbName] [list value $dbName] [list type
java.lang.String] [list required true] [list description "The DB2 database name"]]
set newprops [list $dbname_attr]
set resprops [list resourceProperties $newprops]
set dsProp_attrs [list propertySet [list $resprops]]
#-----
# Set up the attributes for a connection pool
#-----
# ConnectionPool attributes are:
# "agedTimeout Long"
# "connectionTimeout Long"
# "maxConnections Integer"
# "minConnections Integer"
# "purgePolicy ENUM(EntirePool, FailingConnectionOnly)"
# "reapTime Long"
# "unusedTimeout Long"
set agedTimeout_attr [list agedTimeout "0"]
set connectionTimeout_attr [list connectionTimeout "1800"]
set maxConnections_attr [list maxConnections "10"]
set minConnections_attr [list minConnections "1"]
set purgePolicy_attr [list purgePolicy "EntirePool"]
set reapTime_attr [list reapTime "180"]
set unusedTimeout_attr [list unusedTimeout "1800"]
set connPool_attrs [list connectionPool [list $agedTimeout_attr $connectionTimeout_attr
$maxConnections_attr $minConnections_attr $purgePolicy_attr $reapTime_attr
$unusedTimeout_attr]]
#-----
# Set up a relational resource adapter.
#-----
set rra [$AdminConfig getid "/Node:$bn/J2CResourceAdapter:WebSphere Relational Resource
Adapter/"]
set rra_attr [list relationalResourceAdapter $rra]
#-----
# Set up DataSource Attributes
#-----
set name_attr [list name $dsName]
set cat_attr [list category $cat]
set prov_attr [list provider $jdbcProvId]
set jndiName [concat jdbc/$dsName]
set jndiName_attr [list jndiName $jndiName]
set desc_attr [list description $desc]
set authmech_attr [list authMechanismPreference $authMech]
set authDataAlias_attr [list authDataAlias $authAlias]
set authMap_attr [list mapping [list [list authDataAlias $authAlias] [list
mappingConfigAlias DefaultPrincipalMapping]]]
set cache_attr [list statementCacheSize "10"]
set dsHelper_attr [list datasourceHelperClassname $dsHelper]
set ds_attrs [list $name_attr $jndiName_attr $dsHelper_attr $authmech_attr
$authDataAlias_attr $authMap_attr $desc_attr $cat_attr $connPool_attrs $rra_attr
$cache_attr $dsProp_attrs]
puts " "
puts "ConfigProcs: AdminConfig create DataSource $jdbcProvId $ds_attrs"
puts " "
$AdminConfig create DataSource $jdbcProvId $ds_attrs
}

#-----
#

```

```

# save Configuration
#-----
proc saveConfiguration {} {
    global AdminConfig
    puts "\nConfigProcs: Saving the configuration"
    $AdminConfig save
}

#-----
#
# install Enterprise App - this takes the server and the source ear file for the target
application
#                               the appname is only used for naming the application
# return 6 on failure
#-----
proc installEar {server sourceEarFile appname} {
    global AdminApp
    puts "ConfigProcs:  installApp for $server $sourceEarFile $appname"

    if {[catch {$AdminApp install "$sourceEarFile" [subst {-appname $appname -server
$server}]} result]} {
        puts stderr "\nAn error occurred while installing the Enterprise application contained
in"
        puts stderr "$sourceEarFile"
        return -code error -errorcode 6 $result
    } else {
        puts "\nThe Enterprise application was installed successfully"
        puts "In order to use the application please start it first"
    }
}

#-----
#
# install Web App - this takes the server and the source war file for the target
application
#                               appname is only used for naming the application
#                               contextRoot is used to define the URL at which the application is
visible
#                               e.g. for http://localhost:7080/myownurl you would use myownurl for the
context root
# return 5 on failure
#-----
proc installWar {server sourceWarFile appname contextRoot} {
    global AdminApp
    puts "ConfigProcs:  installWar for $server $sourceWarFile $appname $contextRoot"

    if {[catch {$AdminApp install "$sourceWarFile" [subst {-contextroot $contextRoot
-appname $appname -server $server -usedefaultbindings}]} result]} {
        puts stderr "\nAn error occurred while installing the Web application contained in"
        puts stderr "$sourceWarFile"
        return -code error -errorcode 5 $result
    } else {
        puts "\nThe Web application was installed successfully"
        puts "In order to use the application please start it first"
    }
}

#-----

```

```

# installApp - Install the specified application ear file if an
#               application with the same name does not exist.
#-----
proc installApp {appName ear deployejb deployws defaultBindings earMetaData dbType target}
{
    #   appName      - application name
    #   ear          - ear file
    #   deployejb    - deploy ejb (true|false)
    #   deployws     - deploy webservises (true|false)
    #   defaultBindings - use default binding (true|false)
    #   earMetaData  - use MetaData from ear (true|false)
    #   dbType       - ejb deploy db type
    #   target[0]    - node name or cluster name
    #   target[1]    - server name

    global AdminControl
    global AdminApp

    puts ""
    puts "Installing application {${appName}}..."

    # Check if the application already exists

    set appList [AdminApp list]
    foreach item $appList {
        if {[string first $appName $item] >= 0} {
            set app $item
            break
        }
    }

    if {[info exists app]} {
        puts "  Application Name:    ${appName}"
        puts "  Ear file:              ${ear}"
        if {[llength $target] == 1} {
            puts "  Target Cluster:        [lindex $target 0]"
        } else {
            puts "  Target Node:           [lindex $target 0]"
            puts "  Target Server:         [lindex $target 1]"
        }
        puts "  Deploy EJB:            ${deployejb}"
        puts "  Deploy WebServices:    ${deployws}"
        puts "  Use default bindings:  ${defaultBindings}"
        puts "  Use Ear MetaData:      ${earMetaData}"
        puts "  Deployed DB Type:      ${dbType}"

        set parms "-appname $appName"
        if {$deployejb == "true"} {
            append parms " -deployejb"
            append parms " -deployejb.dbtype $dbType"
        }
        if {$deployws == "true"} {
            append parms " -deployws"
        }
        if {$defaultBindings == "true"} {
            append parms " -usedefaultbindings"
        }
        if {$earMetaData == "true"} {
            append parms " -useMetaDataFromBinary yes"
        }
    } else {

```

```

        append parms " -useMetaDataFromBinary no"
    }

    if {[length $target] == 1} {
        append parms " -cluster [lindex $target 0]"
    } else {
        append parms " -node [lindex $target 0] -server [lindex $target 1]"
    }

    set parms1 [subst {$parms}]

    puts "Starting application install..."

    set app [$AdminApp install $ear $parms1]

    puts "Install completed successfully!"
} else {
    puts "${appName} already exists!"
}

return $app
}

#-----
#
# createVirtualHost - this takes base node and a hostname
#
# VirtualHost Attributes are:
# "aliases HostAlias*"
# "mimeTypes MimeEntry*" - we will take the defaults for mimeTypes
# "name String"
#-----
proc createVirtualHost {bn hostName} {
    global AdminConfig
    set vh_parent [$AdminConfig getid /Node:$bn/]
    set vh_name_attr [list name $hostName]
    set vh_attrs [list $vh_name_attr]
    # create the virtual host
    puts " "
    puts "ConfigProcs: AdminConfig create VirtualHost $vh_parent $vh_attrs"
    puts " "
    $AdminConfig create VirtualHost $vh_parent $vh_attrs
}

#-----
#
# createHostAlias - this takes the virtual hostname, and an alias
#                  with the associated port numbers as arguments
#                  It is common to have the alias be *
#
# HostAlias Attributes are:
# "hostname String"
# "port String"
#-----
proc createHostAlias {vHostName alias port} {
    global AdminConfig
    set vhost [$AdminConfig getid /VirtualHost:$vHostName/]
    # set up alias

```

```

        set h_attr [list hostname $alias]
        set p_attr [list port $port]
        set alias_attr [list $h_attr $p_attr]
        set vh_cmdAttrs [list [list aliases [list $alias_attr]]]
        # modify the virtual host, creating the new alias
        puts " "
        puts "ConfigProcs: AdminConfig modify $vhost $vh_cmdAttrs"
        puts " "
        $AdminConfig modify $vhost $vh_cmdAttrs
    }

#-----
#
# createHTTPTransport - this takes base node, the server name to contain
#                       these transports, the port number, and
#                       (optionally) the ssl settings to be used
# NOTE: If ssl settings not provided, sslEnabled will be set to false
#
# HTTPTransport Attributes are:
# "address EndPoint"
# "external Boolean"
# "properties Property(TypedProperty)*"
# "sslConfig String"
# "sslEnabled Boolean"
#-----
proc createHTTPTransport {bn serv host port sslSettings} {
    global AdminConfig

    set parent [$AdminConfig getid /Node:$bn/]

    # Identify the server and assign it to the server variable
    set server [$AdminConfig getid /Node:$bn/Server:$serv/]

    # Identify the Web container belonging to the server and assign it to the wc variable.
    set wc [$AdminConfig list WebContainer $server]

    #Determine if this transport is to be sslEnabled
    if {($sslSettings == "")} {
        set ssl [list sslEnabled false]
    } else {
        set ssl [list sslConfig $sslSettings sslEnabled true]
    }
    # set host-port
    set h_attr [list host $host]
    set p_attr [list port $port]
    set endPoint_attr [list $h_attr $p_attr]

    #Create HTTP Transport
    set transpAddr1 [list [list address $endPoint_attr] $ssl]
    puts " "
    puts "ConfigProcs: AdminConfig create HTTPTransport $wc $transpAddr1"
    puts " "
    $AdminConfig create HTTPTransport $wc $transpAddr1
}

#
# dumpStack - a utility method to dump information about the error condition
# Parameters
# result - result of the previous command, just a string.

```



```

#
# typical use
# if {[catch {my command} result]} {
#     dumpStack $result
#     <recover from error or exit>
# }
proc dumpStack {result} {
    global errorInfo
    global errorCode
    puts stderr "ConfigProcs:
    _____Trace_____ "
    puts stderr "[clock format [clock seconds]]"
    puts stderr "Error Code=$errorCode"
    puts stderr $result
    puts stderr $errorInfo
}

# Start an installed application
proc startApp { myApplication server } {
    global AdminControl
    puts "ConfigProcs: startApp $myApplication $server"
    set appManager [AdminControl queryNames type=ApplicationManager,process=$server,*]
    $AdminControl invoke $appManager startApplication $myApplication
}

# Stop an installed application
proc stopApp { myApplication server } {
    global AdminControl
    puts "ConfigProcs: stopApp $myApplication $server"
    set appManager [AdminControl queryNames type=ApplicationManager,process=$server,*]
    $AdminControl invoke $appManager stopApplication $myApplication
}

# getMessage - a procedure that extracts the NLS message in the same resource bundle
#               as that used by the other java parts in the sample.
#
# parameters - key - a key defined in NLSKeys that will be used to look up the message in
MessagesNLS /_en /_fr etc
#               - args - variable number of arguments that can be provided as replacement
parameters
#               e.g. if key = COPYFILE_FAIL , message = IRU00004: Copy file {0} to
{1} failed
#               and you invoke getMessage COPYFILE_FAIL gaga baba, you would
get back
#               IRU00004: Copy file gaga to baba failed
#
# Several advanced techniques are used here so please be careful
#
# the java classes must be in the classpath. You can see the CP by invoking this in wsadmin
# wsadmin> foreach {key value} [array get env] { puts $key=$value }
# set the classpath like this
# wsadmin -wsadmin_classpath "C:\Program
Files\IBM\MidMarketRuntime\SolutionEnabler\unpacked"
#
# java::new to create a new object as in
# set x [java::new String "abcd"]

```

```

# java::call to call a static method as
# set abs_x [java::call Math.abs x]
# java::field get the value of a class or instance variable as in
# set max_int [java::field Integer MAX_VALUE]
# java::null for setting the null value as in
# set x [java::null]
# java::isnull tests for null as in
# if {[java::isnull $x]} ...
#
proc getMessage {key args} { ;# args is a variable argument list
    if {[string first "Linux" $env(os.name)] == 0} {
        set msgkey [java::field com.ibm.iru.samplelnx.NLSKeys $key]
        set defaultBundle [java::call java.util.ResourceBundle getBundle
com.ibm.iru.samplelnx.MessagesNLS]
    } else {
        set msgkey [java::field com.ibm.iru.samplewin.NLSKeys $key]
        set defaultBundle [java::call java.util.ResourceBundle getBundle
com.ibm.iru.samplewin.MessagesNLS]
    }
    set jnk [$defaultBundle getString $msgkey]
    set attrs [java::new {Object[] } [llength $args] $args ]
    set msg [java::call java.text.MessageFormat format $jnk $attrs]
    return -code ok $msg
}

# Modify Variable Map entries e.g. DB2_JDBC_DRIVER_PATH
# This should be used for classpath for the JDBC provider instead of a direct path to the
# driver. Procedure for later releases.
#
# $AdminConfig showall $varMap
#
proc updateVariableMap {key newValue serverName nodeName} {

    puts "ConfigProc: updateVariableMap $key $newValue $serverName $nodeName"
    global AdminConfig
    # setting WebSphere variables at node level
    set varMap [$AdminConfig getid /Node:$nodeName/VariableMap:/]
    set subst [lindex [$AdminConfig showAttribute $varMap entries] 0]

    set foundit "false"

    foreach sub $subst {
        set varName [$AdminConfig showAttribute $sub symbolicName]
        if {$varName == $key} {
            if {$newValue != ""} {
                puts "Setting $key to $newValue"
                $AdminConfig modify $sub [subst {{value "$newValue"}}]
                set foundit "true"
                break
            }
        }
    }

    #let us create one if none found
    if {$foundit == "false"} {
        # create a list of items
        set nameattr1 [list symbolicName $key]
        set valattr1 [list value $newValue]
        set attr1 [list $nameattr1 $valattr1]
    }
}

```

```

        set attrs [list $attr1]

        puts "creating $key with value of $newValue"
        $AdminConfig modify $varMap [subst {{entries {$attrs}}}]
    }
}

# securedb2 - this is a way to allow WebSphere to access DB2 without having to source the
# db2profile
#         you only need to do this if sourcing the db2profile script in your .profile
#         is not acceptable to you.
# parameters
# db2home - installed location of db2 e.g. /opt/IBM/db2/V8.1
# instance - the db2instance to be used e.g. db2inst1
# server - your server name - optional - defaults to server1 if not provided
#
#
# All you need to do is get the db2 environment variables configured for the JVM .
# It is the only way for the Type 2 driver to get to the native libraries.
# The DB2INSTANCE variable will probably always have to be set, but if you are using ONLY
# the new
# Type 4 driver then you don't have to set LD_LIBRARY_PATH or LIBPATH as the native
# libraries aren't needed.
#
# The following 3 variables will be added to the java invocation of the app server
#
# LD_LIBRARY_PATH=/home/db2inst1/sqllib/lib:/home/db2inst1/sqllib/bin
# LIBPATH=/home/db2inst1/sqllib/lib:/home/db2inst1/sqllib/bin
# DB2INSTANCE=db2inst1 (or whatever your DB2 instance user name is).

proc securedb2 {bn db2home instance {server server1}} {
    puts "ConfigProcs: securedb2 $bn $db2home $instance $server"
    global AdminConfig
    set appServer [$AdminConfig getid /Node:$bn/Server:$server/]
    #set jvm [$AdminConfig list JavaVirtualMachine $appServer]
    # $AdminConfig modify $jvm $attr
    set attr [subst {{environment {{{name "DB2INSTANCE"}}{required false}{value
"$instance"}} {{name "LD_LIBRARY_PATH"}}{required false}{value "$db2home/lib"}} {{name
"LIBPATH"}}{required false}{value "$db2home/lib"}}}}}]]
    set process [$AdminConfig list ProcessDef $appServer]
    $AdminConfig modify $process $attr
    foreach variableSubstitution [$AdminConfig list VariableSubstitutionEntry] {
        foreach varSubstitutionEntry [$AdminConfig showall $variableSubstitution] {
            if { [lsearch -regexp $varSubstitutionEntry "DB2_JDBC_DRIVER_PATH"] >= 0 } {
                set valueVar [list value "$db2home/java"]
                set varSubAttribs [list $valueVar]
                $AdminConfig modify $variableSubstitution $varSubAttribs
                break
            }
        }
    }
}

#-----
# createSIBus - Create a new SIBus if one does not exist. Otherwise,
#               return the existing SIBus.
#-----

```

```

proc createSIBus {busName authAlias} {
    #    busName    - SIBus name
    #    authAlias  - authentication alias name

    global AdminTask
    global AdminConfig

    puts " "
    puts "Creating SIBus ${busName}..."

    # Check if the SIBus already exists

    set SIBus [AdminConfig getid "/SIBus:${busName}/"]

    if {$SIBus == ""} {
        set parms [list -bus $busName -interEngineAuthAlias $authAlias]
        if {[catch {set SIBus [AdminTask createSIBus $parms]} result]} {
            puts "WSADMIN EXCEPTION: ${result}"
            puts "Terminating due to exception!"
            exit
        } else {
            puts "${busName} created successfully!"
        }
    } else {
        puts "${busName} already exists!"
    }

    return $busName
}

#-----
# addSIBusMember - Add the specified server or cluster to the
#                  SIBus if it does not already exist. Assumes that the
#                  specified SIBus already exists.
#-----
proc addSIBusMember {busName defaultDS dsJndi optArgs} {
    #    busName    - SIBus name
    #    defaultDS  - create default DS (true|false)
    #    dsJndi     - jndi name of the datasource (only used if defaultDS = false)
    #    optArgs[0] - cluster name or node name
    #    optArgs[1] - server name

    global AdminTask
    global AdminConfig

    puts " "
    if {[length $optArgs] == 1} {
        set clusterName [lindex $optArgs 0]
        puts "Adding SIBus member ${clusterName}..."
    } else {
        set clusterName "none"
        set nodeName    [lindex $optArgs 0]
        set serverName   [lindex $optArgs 1]
        puts "Adding SIBus member ${nodeName} - ${serverName}..."
    }

    puts " Default DataSource:    ${defaultDS}"
    if {$defaultDS == "false"} {

```

```

    puts " Datasource JNDI Name: ${dsJndi}"
}

# Check if the bus member already exists

set parms [list -bus $busName]
set busMembers [$AdminTask listSIBusMembers $parms]

foreach item $busMembers {
    set cluster [$AdminConfig showAttribute $item cluster]
    set node [$AdminConfig showAttribute $item node]
    set server [$AdminConfig showAttribute $item server]

    if {$cluster == $clusterName || ($server == $serverName && $node == $nodeName)} {
        set member $item
        break
    }
}

if {[llength $optArgs] == 1} {
    set parms [list -bus $busName -cluster $clusterName -createDefaultDatasource
$defaultDS]
} else {
    set parms [list -bus $busName -node $nodeName -server $serverName
-createDefaultDatasource $defaultDS]
}

if {$defaultDS == "false"} {
    lappend parms -datasourceJndiName $dsJndi
}

if {[info exists member]} {
    if {[catch {set member [$AdminTask addSIBusMember $parms]} result]} {
        puts "WSADMIN EXCEPTION: ${result}"
        puts "Terminating due to exception!"
        exit
    } else {
        puts "SIBus member added successfully!"
    }
} else {
    puts "Bus member already exists!"
}

return $member
}

#-----
# createSIBDestination - Create a new SIB Destination if one with the same
# name does not exist on the specified SIBus. Otherwise,
# return the existing Destination.
#-----
proc createSIBDestination {SIBus destName destType reliability optArgs} {
    # SIBus - SIBus name
    # destName - destination name
    # destType - destination type
    # reliability - reliability
    # optArgs[0] - cluster name or node name
    # optArgs[1] - server name

global AdminTask

```

```

global AdminConfig

if {[length $optArgs] == 1} {
    set clusterName [lindex $optArgs 0]
} else {
    set nodeName    [lindex $optArgs 0]
    set serverName  [lindex $optArgs 1]
}

puts " "
puts "Creating SIB Destination ${destName}..."

# Check if the SIB Destination already exists

set parms [list -bus $SIBus]
set destList [$AdminTask listSIBDestinations $parms]

foreach item $destList {
    set ident [$AdminConfig showAttribute $item identifier]
    if {$ident == $destName} {
        set dest $item
        break
    }
}

if {[info exists dest]} {
    puts " Destination Name:  ${destName}"
    puts " Destination Type:  ${destType}"
    puts " Reliability:       ${reliability}"
    if {$destType == "Queue"} {
        if {[length $optArgs] == 1} {
            puts " Cluster Name:    ${clusterName}"
        } else {
            puts " Node Name:      ${nodeName}"
            puts " Server Name:    ${serverName}"
        }
    }
}

set parms [list -bus $SIBus -name $destName -type $destType -reliability
$reliability]
if {$destType == "Queue" && [length $optArgs] == 1} {
    lappend parms -cluster $clusterName
} elseif {$destType == "Queue"} {
    lappend parms -node $nodeName -server $serverName
}

if {[catch {set dest [$AdminTask createSIBDestination $parms]} result]} {
    puts "WSADMIN EXCEPTION: ${result}"
    puts "Terminating due to exception!"
    exit
} else {
    puts "${destName} created successfully!"
}
} else {
    puts "$destName already exists!"
}

return $dest
}

```

```

#-----
# createJMSConnectionFactory - Create a new JMS Connection Factory
#                               if one with the same name does not exist on the SIBus.
#                               Otherwise, return the existing Connection Factory.
#-----
proc createJMSConnectionFactory {SIBus cfName cfType jndiName authAlias scope} {
    # Create JMS Connection Factory
    #   SIBus      - SIBus name
    #   cfName     - connection factory name
    #   cfType     - connection factory type
    #   jndiName   - connection factory jndi name
    #   authAlias  - authentication alias name
    #   scope      - scope

    global AdminTask

    puts " "
    puts "Creating JMS ${cfType} Connection Factory ${cfName}..."

    # Check if the connection factory already exists

    set parms [list -type $cfType]
    set cfList [$AdminTask listSIBJMSConnectionFactories $scope $parms]

    foreach item $cfList {
        if {[string first $cfName $item] >= 0} {
            set connectionFactory $item
            break
        }
    }

    if {[info exists connectionFactory]} {
        puts " Connection Factory Name:  ${cfName}"
        puts " Connection Factory Type:  ${cfType}"
        puts " JNDI Name:                  ${jndiName}"

        set parms [list -name $cfName -jndiName $jndiName -busName $SIBus -type $cfType
            -authDataAlias $authAlias]
        if {[catch {set connectionFactory [$AdminTask createSIBJMSConnectionFactory $scope
            $parms]} result]} {
            puts "WSADMIN EXCEPTION: ${result}"
            puts "Terminating due to exception!"
            exit
        } else {
            puts "${cfName} created successfully!"
        }
    } else {
        puts "$cfName already exists!"
    }

    return $connectionFactory
}

#-----
# createJMSQueue - Create a new JMS Queue if one with the same
#                 name does not exist at the specified scope. Otherwise,
#                 return the existing JMS Queue.

```

```

#-----
proc createJMSQueue {qName jndiName SIBDest delMode scope} {
    #    qName    - queue name
    #    jndiName - queue jndi name
    #    SIBDest  - SIB destination
    #    delMode  - delivery mode
    #    scope    - scope

    global AdminTask

    puts " "
    puts "Creating JMS Queue ${qName}..."

    # Check if the queue already exists

    set qList [AdminTask listSIBJMSQueues $scope]
    foreach item $qList {
        if {[string first $qName $item] >= 0} {
            set queue $item
            break
        }
    }

    if {[info exists queue]} {
        puts " Queue Name:      ${qName}"
        puts " JNDI Name:         ${jndiName}"
        puts " SIB Destination:    ${SIBDest}"
        puts " Delivery Mode:      ${delMode}"

        set params [list -name $qName -jndiName $jndiName -queueName $SIBDest -deliveryMode
$delMode]
        if {[catch {set queue [AdminTask createSIBJMSQueue $scope $params]} result]} {
            puts "WSADMIN EXCEPTION: ${result}"
            puts "Terminating due to exception!"
            exit
        } else {
            puts "${qName} created successfully!"
        }
    } else {
        puts "$qName already exists!"
    }

    return $queue
}

#-----
# createJMSTopic - Create a new JMS Topic if one with the same
#                  name does not exist at the specified scope. Otherwise,
#                  return the existing JMS Topic.
#-----
proc createJMSTopic {tName jndiName tSpace delMode scope} {
    #    tName    - topic name
    #    jndiName - topic jndi name
    #    tSpace    - topic space
    #    delMode   - delivery mode
    #    scope     - scope

    global AdminTask

```



```

puts " "
puts "Creating JMS Topic ${tName}..."

# Check if the topic already exists

set tList [$AdminTask listSIBJMSTopics $scope]
foreach item $tList {
    if {[string first $tName $item] >= 0} {
        set topic $item
        break
    }
}

if {[info exists topic]} {
    puts " Topic Name:      ${tName}"
    puts " JNDI Name:       ${jndiName}"
    puts " Topic Space:       ${tSpace}"
    puts " Delivery Mode:     ${delMode}"

    set params [list -name $tName -jndiName $jndiName -topicName $tName -topicSpace
$tSpace -deliveryMode $delMode]
    if {[catch {set topic [$AdminTask createSIBJMSTopic $scope $params]} result]} {
        puts "WSADMIN EXCEPTION: ${result}"
        puts "Terminating due to exception!"
        exit
    } else {
        puts "${tName} created successfully!"
    }
} else {
    puts "$tName already exists!"
}

return $topic
}

#-----
# createMDBActivationSpec - Create a new MDB Activation Spec if one
# with the same name does not exist at the specified
# scope. Otherwise, return the existing Activation Spec.
#-----
proc createMDBActivationSpec {mdbName jndiName SIBus JMSDestJndi destType authAlias scope}
{
    # mdbName - MDB name
    # jndiName - activation spec jndi name
    # SIBus - SIBus name
    # JMSDestJndi - JMS destination JNDI name
    # destType - destination type
    # authAlias - authentication alias name
    # scope - scope

    global AdminTask

    puts " "
    puts "Creating MDB Activation Spec ${mdbName}..."

    # Check if the activation spec already exists

    set asList [$AdminTask listSIBJMSActivationSpecs $scope]
    foreach item $asList {

```

```

        if {[string first $mdbName $item] >= 0} {
            set mdb $item
            break
        }
    }

    if ![info exists mdb] {
        puts "  MDB Activation Spec Name:   ${mdbName}"
        puts "  JNDI Name:                       ${jndiName}"
        puts "  JMS Destination JNDI Name:  ${JMSDestJndi}"
        puts "  Destination Type:           ${destType}"

        set params [list -name $mdbName -jndiName $jndiName -busName $SIBus
        -destinationJndiName $JMSDestJndi -destinationType $destType -authenticationAlias
        $authAlias]
        if {[catch {set mdb [$AdminTask createSIBJMSActivationSpec $scope $params]} result]}
        {
            puts "WSADMIN EXCEPTION: ${result}"
            puts "Terminating due to exception!"
            exit
        } else {
            puts "${mdbName} created successfully!"
        }
    } else {
        puts "$mdbName already exists!"
    }

    return $mdb
}

#-----
# createJDBCProvider - Create a new JDBC Provider if one with the
#                      same name does not exist in the specified scope. Otherwise,
#                      return the existing JDBCProvider. The 3 types or providers
#                      currently supported include DB2 JCC, DB2 CLI, and Oracle.
#-----
proc createJDBCProvider {providerType path XA scope} {
    # providerType - provider type (db2|db2cli|oracle|cloudscape)
    # path          - driver classpath path
    # XA            - XA (true|false)
    # scope         - scope

    global AdminConfig

    # Determine properties based on providerType

    if ${providerType} == "db2" {
        if ${XA} == "true" {
            set providerName          "DB2 Universal JDBC Driver Provider (XA)"
            set implementationClassName "com.ibm.db2.jcc.DB2XADataSource"
        } else {
            set providerName          "DB2 Universal JDBC Driver Provider"
            set implementationClassName "com.ibm.db2.jcc.DB2ConnectionPoolDataSource"
        }
    } elseif ${providerType} == "oracle" {
        if ${XA} == "true" {
            set providerName          "Oracle JDBC Driver (XA)"
            set implementationClassName "oracle.jdbc.xa.client.OracleXADataSource"
        } else {

```

```

        set providerName          "Oracle JDBC Driver"
        set implementationClassName "oracle.jdbc.pool.OracleConnectionPoolDataSource"
    }
} elseif (${providerType} == "db2cli") {
    if (${XA} == "true") {
        set providerName          "DB2 Legacy CLI-based Type 2 JDBC Driver (XA)"
        set implementationClassName "COM.ibm.db2.jdbc.DB2XADataSource"
    } else {
        set providerName          "DB2 Legacy CLI-based Type 2 JDBC Driver"
        set implementationClassName "COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource"
    }
} elseif (${providerType} == "cloudscape") {
    if (${XA} == "true") {
        set providerName          "Cloudscape JDBC Provider (XA)"
        set implementationClassName "com.ibm.db2j.jdbc.DB2jXADataSource"
    } else {
        set providerName          "Cloudscape JDBC Provider"
        set implementationClassName "com.ibm.db2j.jdbc.DB2jConnectionPoolDataSource"
    }
}

puts " "
puts "Creating JDBC Provider ${providerName}..."

# Check if the JDBC provider already exists

set name [getName $scope]
set stIndex [expr [string first "|" $scope] + 1]
set endIndex [expr [string first "." $scope] - 1]
set type [string range $scope $stIndex $endIndex]

if (${type} == "cell") {
    set provider [$AdminConfig getid "/Cell:$name/JDBCProvider:\${providerName}/"]
} elseif (${type} == "node") {
    set provider [$AdminConfig getid "/Node:$name/JDBCProvider:\${providerName}/"]
} elseif (${type} == "server") {
    set provider [$AdminConfig getid "/Server:$name/JDBCProvider:\${providerName}/"]
}

if {$provider == ""} {
    puts " Provider Name:      ${providerName}"
    puts " Implementation Class: ${implementationClassName}"
    puts " XA enabled:             ${XA}"

    set attrs [subst {{classpath "$path"} {implementationClassName
$implementationClassName} {name "${providerName}" {providerType "${providerName}"
{description "${providerName}" {xa "${XA}}}}}
    if {[catch {set provider [$AdminConfig create JDBCProvider $scope $attrs]} result]} {
        puts "WSADMIN EXCEPTION: ${result}"
        puts "Terminating due to exception!"
        exit
    } else {
        puts "${providerName} created successfully!"
    }
} else {
    puts "${providerName} already exists!"
}

return $provider
}

```

```

#-----
# createDB2orCloudscapeDatasource - Create a new Datasource if one
# with the same name does not exist at the specified
# scope. Otherwise, return the existing Datasource.
#-----
proc createDB2orCloudscapeDatasource {datasourceName jndiName stmtCacheSz provider
providerType authAliasName description scope dbName optArgs} {
    # datasourceName - Datasource name
    # jndiName - JNDI name
    # stmtCacheSz - Statement Cache Size
    # provider - provider
    # providerType - provider type (db2|db2cli|cloudscape)
    # authAliasName - JAAS authentication alias name
    # description - description
    # scope - scope
    # dbName - dbName (for DB2 and Cloudscape)
    # optArgs[0] (jccType) - JDBC driver type (2|4)
    # optArgs[1] (hostname) - database hostname (for type 4 driver)
    # optArgs[2] (port) - port number (for type 4 driver)

    set jccType [lindex $optArgs 0]
    set hostname [lindex $optArgs 1]
    set port [lindex $optArgs 2]

    global AdminConfig

    # Connection pool properties

    set connectionTimeout 1800
    set maxConnections 50
    set minConnections 1
    set reapTime 180
    set unusedTimeout 1800
    set agedTimeout 0

    if ${providerType} == "db2" {
        set datasourceHelperClassname
"com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper"
    } elseif ${providerType} == "db2cli" {
        set datasourceHelperClassname "com.ibm.websphere.rsadapter.DB2DataStoreHelper"
    } elseif ${providerType} == "cloudscape" {
        set datasourceHelperClassname
"com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper"
    }

    set name [getName $scope]
    set stIndex [expr [string first "|" $scope] + 1]
    set endIndex [expr [string first "." $scope] - 1]
    set type [string range $scope $stIndex $endIndex]

    if ${type} == "cell" {
        set radapter [$AdminConfig getid "/Cell:$name/J2CResourceAdapter:WebSphere Relational
Resource Adapter/"]
    } elseif ${type} == "node" {
        set radapter [$AdminConfig getid "/Node:$name/J2CResourceAdapter:WebSphere Relational
Resource Adapter/"]
    } elseif ${type} == "server" {
        set radapter [$AdminConfig getid "/Server:$name/J2CResourceAdapter:WebSphere
Relational Resource Adapter/"]
    }
}

```

```

}

puts " "
puts "Creating DataSource ${datasourceName}..."

# Check if the DataSource already exists

foreach item [$AdminConfig list DataSource $scope] {
    set tmpProvider [$AdminConfig showAttribute $item provider]
    if {[string first $datasourceName $item] >= 0 && [string first $provider
$tmpProvider] >= 0} {
        set datasource $item
        break
    } elseif {[string first $datasourceName $item] >= 0} {
        puts "${datasourceName} already exists in another JDBC Provider!"
        puts "Please rename or delete the existing DataSource before proceeding."
        exit
    }
}

# If DataSource does not yet exists, create a new one

if {[info exists datasource]} {
    puts "  Datasource Name:      ${datasourceName}"
    puts "  JNDI Name:              ${jndiName}"
    puts "  Statement Cache Size:    ${stmtCacheSz}"
    puts "  Database Name:           ${dbName}"
    if {$providerType == "db2"} {
        puts "    JDBC Driver Type:      ${jccType}"
        if {$jccType == 4} {
            puts "      Hostname:            ${hostname}"
            puts "      Port Number:         ${port}"
        }
    }

    if {!($providerType == "cloudscape")} {
        set attrs [subst {{name $datasourceName} {description "$description"} {jndiName
$jndiName} {statementCacheSize $stmtCacheSz} {authDataAlias $authAliasName}
{datasourceHelperClassname $datasourceHelperClassname} {authDataAlias $authAliasName}
{xaRecoveryAuthAlias "\""} {providerType "[AdminConfig showAttribute $provider
providerType]"}}}
    } else {
        set attrs [subst {{name $datasourceName} {description "$description"} {jndiName
$jndiName} {statementCacheSize $stmtCacheSz} {datasourceHelperClassname
$datasourceHelperClassname} {xaRecoveryAuthAlias "\""} {providerType "[AdminConfig
showAttribute $provider providerType]"}}}
    }

    if {[catch {set datasource [AdminConfig create DataSource $provider $attrs]}
result]} {
        puts "WSADMIN EXCEPTION: ${result}"
        puts "Terminating due to exception!"
        exit
    } else {
        #Create the datasource properties...

        puts ""
        puts "  Creating Datasource properties..."
    }
}

```

```

set propSet [$AdminConfig create J2EEResourcePropertySet $datasource {}]
set attrs [subst {{name databaseName} {type java.lang.String} {value $dbName}}]
$AdminConfig create J2EEResourceProperty $propSet $attrs

if {{providerType} == "db2"} {
set attrs [subst {{name driverType} {type java.lang.Integer} {value $jccType}}]
$AdminConfig create J2EEResourceProperty $propSet $attrs

if {{jccType} == 4} {
set attrs [subst {{name serverName} {type java.lang.String} {value
$hostname}}]
$AdminConfig create J2EEResourceProperty $propSet $attrs
set attrs [subst {{name portNumber} {type java.lang.Integer} {value $port}}]
$AdminConfig create J2EEResourceProperty $propSet $attrs
}
}

#Create the connection pool object...
puts " Creating Connection Pool using defaults..."
set attrs [subst {{connectionTimeout $connectionTimeout} {maxConnections
$maxConnections} {minConnections $minConnections} {reapTime $reapTime} {unusedTimeout
$unusedTimeout} {agedTimeout $agedTimeout}}]
$AdminConfig create ConnectionPool $datasource $attrs

#Create the connection factory
puts " Creating Connection Factory..."
set cfName $datasourceName
append cfName "_CF"

if {!($providerType == "cloudscape")} {
set attrs [subst {{name ${cfName}} {authMechanismPreference BASIC_PASSWORD}
{cmpDatasource $datasource} {authDataAlias $authAliasName}}]
} else {
set attrs [subst {{name ${cfName}} {authMechanismPreference BASIC_PASSWORD}
{cmpDatasource $datasource}}]
}

$AdminConfig create CMPConnectionFactory $radapter $attrs

puts "${datasourceName} created successfully!"
}
} else {
puts "${datasourceName} already exists in current JDBC Provider!"
}
}

return $datasource
}

#-----
# getName - Return the base name of the config object.
#-----
proc getName {args} {
# arg[0] - object id

set id [lindex $args 0]
set endIndex [expr [string first "(" $id] - 1]

return [string range $id 0 $endIndex]
}

```

```

}

# Remove everything the sample setup
# Assuming nobody changed the script file
# Utility proc maily for test
#
# Typical invocation
# cleanup RuntimeDocumentMgmt DB2JDBCProv1 DB2AuthAlias
#
proc cleanup { {appName RuntimeDocumentMgmt} {jdbcProviderName DB2JDBCProv1} {jaasAliasName
DB2AuthAlias}} {
    global AdminApp
    global AdminConfig
    if {[catch {
        stopApp $appName
    } result]} { ;#ignore it }
    if {[catch {
        $AdminApp uninstall $appName
    } result]} { ;#ignore it }

    if {[catch {
        $AdminConfig remove [$AdminConfig getid /JDBCProvider:$jdbcProviderName/]
    } result]} { ;#ignore it }

    if {[catch {
        foreach i [$AdminConfig list JAASAuthData] {
            if { [$AdminConfig showAttribute $i alias] == $jaasAliasName } {
                $AdminConfig remove $i
            }
        }
    } result]} { ;#ignore it }

    $AdminConfig save
}

```

The WebSphereScript.jacl script file

Example B-8 is a sample WebSphereScript.jacl file which was used for deploying the Trade6 application.

Example: B-8 Source of WebSphereScript.jacl

```

# %I% %W% %G% %U%
# *****
# *****
# ***** Main routine *****
# ***** Get input values, verify all required values exist *****
# *****
# *****

puts " "
puts "Beginning of Script main"
puts "The number of passed arguments = $argc "

#-----
# This Script requires the following input parameters:
# 1 - directory where the scripts are located

```

```

# 2 - fully qualified name of response file
#-----
if {$argc != 2} {
    puts " "
    puts " This Script requires the following input parameters:"
    puts " 1 - the scripts directory"
    puts " 2 - fully qualified name of properties file"
    puts " "
    exit
} else {
    puts "Scripts dir = [lindex $argv 0] "
    puts "Properties File = [lindex $argv 1] "
}

#-----
# Get the input parms
#-----
set scriptsDir [lindex $argv 0]
set respFile [lindex $argv 1]

#-----
# Set source to point to the WebSphereConfigProcs
#-----
set setupScript [eval file join "$scriptsDir SetupProcs.jacl"]
set WasProcsScript [eval file join "$scriptsDir WebSphereConfigProcs.jacl"]

source $setupScript
source $WasProcsScript

#-----
# Use the Connected Node
#-----
global baseNode
set baseNode [$AdminControl getNode];# Get the one and only connected node
puts "base Node = $baseNode"
#-----
# Get the variables values from the response file
#-----

set props [loadProperties $respFile];# load properties from the response file

set DB2Deploy          "DB2UDB_V82"

#-----
# JMS (Messaging) Config Parameters
#-----

#set reliability        "ASSURED_PERSISTENT"
set reliability        "EXPRESS_NONPERSISTENT"

#set deliveryMode       "Persistent"
set deliveryMode       "NonPersistent"

# Queue/Topic Names
set brokerSIBDest      "TradeBrokerJSD"
set topicSpace         "Trade.Topic.Space"
set brokerJMSQCF       "TradeBrokerQCF"
set streamerJMSTCF     "TradeStreamerTCF"
set brokerQueue        "TradeBrokerQueue"

```



```

set streamerTopic      "TradeStreamerTopic"
set brokerMDB          "TradeBrokerMDB"
set streamerMDB        "TradeStreamerMDB"

# *****
# *****
# ***** Start of Configuratin Steps *****
# *****
# *****

#-----
# Set scope - id of scope (cell, node, or server)
#-----

set nodeScope [AdminConfig getid /Node:$baseNode/]
set serverScope [AdminConfig getid /Node:$baseNode/Server:$wasServerName]

#-----
# Create a JAAS authorization alias ( required for V5 DataSources)
#-----

if {[catch {
    createJAASDataAuth $authAliasName $db2UserId $db2Password $authAliasDesc
} result]} {
    # this is an unrecoverable error. Stack should have enough information to rectify
    dumpStack $result
    exit
}

#-----
# Create DB2 JDBCProvider
#-----

if {[catch {
    puts "providerName = $JDBCProviderName classpath = $JDBCclasspath baseNode = $baseNode
wasServerName = $wasServerName "
    createJDBCProviderUsingTemplate $JDBCProviderName $JDBCclasspath $baseNode
$wasServerName
    # createJDBCProvider db2 $JDBCclasspath "true" $nodeScope

} result]} {
    # this is an unrecoverable error. Stack should have enough information to rectify
    dumpStack $result
    exit
}

#-----
# Create DB2 DataSource
#-----

set authMech "BASIC_PASSWORD"
set dsHelperClass "com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper"

    puts "created datasource "
if {[catch {
    # createdB2DataSource $baseNode $wasServerName $JDBCProviderName $datasourceName
    $databaseName $dsHelperClass $authAliasName $authMech $datasourceCategory $datasourceDesc

```

```

set jdbcProvId [AdminConfig getid /JDBCProvider:$JDBCProviderName/]
puts "jdbc provideer id = "
puts $jdbcProvId
puts "datasource name = "
puts $datasourceName
puts "jdbc data "
puts "jdbc/${datasourceName}"
puts "db2"
puts db2
puts "auth alias name"
puts $authAliasName
puts "node scope"
puts $nodeScope
puts "database name"
puts $databaseName
puts "{2}"
puts {2}
puts "$datasourceName jdbc/$datasourceName 60 $jdbcProvId db2 $authAliasName Trade6
Datasource $nodeScope $databaseName [list 4 localhost 50001]"
createDB2orCloudscapeDatasource $datasourceName "jdbc/${datasourceName}" 60 $jdbcProvId
db2 $authAliasName "Trade6 Datasource" $nodeScope $databaseName [list 4 localhost 50001]
} result}} {
# this is an unrecoverable error. Stack should have enough information to rectify
dumpStack $result
exit
}

#-----
# Create the JMS config objects
#-----

puts ""
puts "-----"
puts " Configuring JMS Resources"
puts "-----"

createJAASDataAuth $JMSAuthAliasName $DefaultOSUser $DefaultOSPasswd $JMSAuthAliasDesc
createJAASDataAuth $JMSAuthAliasName $DefaultOSUser $DefaultOSPasswd $JMSAuthAliasDesc

set SIBusName [createSIBus $baseNode $JMSAuthAliasName]
set target [subst {$baseNode $wasServerName}]
addSIBusMember $SIBusName "true" "dummy" $target

# Create the Trade Broker Queue and Trade TopicSpace Destinations

createSIBDestination $SIBusName $brokerSIBDest "Queue" $reliability $target
createSIBDestination $SIBusName $topicSpace "TopicSpace" $reliability [subst {}]

createJMSConnectionFactory $SIBusName $brokerJMSQCF "Queue" "jms/$brokerJMSQCF"
$JMSAuthAliasName $nodeScope
createJMSConnectionFactory $SIBusName $streamerJMSTCF "Topic" "jms/$streamerJMSTCF"
$JMSAuthAliasName $nodeScope

createJMSQueue $brokerQueue "jms/$brokerQueue" $brokerSIBDest $deliveryMode $nodeScope
createJMSTopic $streamerTopic "jms/$streamerTopic" $topicSpace $deliveryMode $nodeScope

```

```

    createMDBActivationSpec $brokerMDB "eis/$brokerMDB" $SIBusName "jms/$brokerQueue"
"javax.jms.Queue" $JMSauthAliasName $nodeScope
    createMDBActivationSpec $streamerMDB "eis/$streamerMDB" $SIBusName "jms/$streamerTopic"
"javax.jms.Topic" $JMSauthAliasName $nodeScope

    puts ""
    puts "-----"
    puts " JMS Resource Configuration Completed!!!"
    puts "-----"

#-----
# Install the Application
# If the application install file is an EAR file then use installEar
# If the application install file is a WAR file then use installWar and include the
contextroot
#-----
puts "Install EAR File"
# the ear file was placed into the WAS installable apps directory
set qualifiedAppInstallFile [eval file join "$installableAppsDir $appInstallFile"]
if {[catch {
    # installEar $wasServerName $qualifiedAppInstallFile $appName $contextRoot
    installApp $appName $qualifiedAppInstallFile "true" "true" "true" "true" $DB2Deploy
$target
} result]} {
    # this is an unrecoverable error. Stack should have enough information to rectify
    dumpStack $result
    exit
}

#-----
# IMPORTANT
# Save the configuration changes made
#-----
saveConfiguration

# Start the application to make the URL active
# Note that an invocation of this script with the "-conntype NONE" parameter will
# cause this procedure to generate an error
startApp $appName $wasServerName

# special Linux section
if {[string first "Linux" $env(os.name)] == 0} {
    # Have to add DB2 environment variables to the server process definition
    # This is done to avoid having to source the db2profile
    # we want to strip the path down to db2 home
    # e.g. /opt/IBM/db2/V8.1/java/db2java.zip becomes /opt/IBM/db2/V8.1
    # securedB2 [file dirname [file dirname $JDBCclasspath]] $db2instance
}

```

```
saveConfiguration
```

```
# *****  
# End Main routine  
# *****
```

The DB2Script.sh script file

Example B-9 is a sample DB2Script.sh file which was used for deploying the Trade6 application.

Example: B-9 DB2Script.sh file

```
#!/bin/bash  
# Licensed Materials - Property of IBM  
#  
# 5724-F71 5724-J10  
#  
# (C) Copyright IBM Corporation 2004, 2005 All Rights Reserved  
#  
# US Government Users Restricted Rights- Use, duplication or  
# disclosure restricted by GSA ADP Schedule Contract with IBM  
# Corp.  
#  
# This is sample code made available for use in accordance with  
# terms set forth in the license agreement document for the IBM  
# Express Runtime.  
#  
# This script creates the database for the Express Runtime sample for Linux.  
# Usage: [Database Name] [Fully qualified script directory] [Document Location]  
# This script must be run by the db2 instance user  
# call su - [db2InstanceUserId] -c "IRU_DB2Script.sh [Database Name] [Script dir]  
[Documents dir]".  
# Note: directory parameters should not have the forward slash at the end  
#echo "DB2Script -this script requires 3 parameters: the database name, the full path to  
the scripts directory, the full path to the documents directory"  
  
if [ "$4" != "" ]  
then  
    DatabaseName=$1  
    ScriptDirectory=$2  
    db2userid=$3  
    db2password=$4  
  
    # Start the db2 database manager if it isn't already started  
    # su - $db2userid -c "db2 db2start"  
    #--  
    #-- cataloging is necessary to recover from a previous installation's left over  
database  
    #-- please ignore any error messages emitted  
    #-- we also do an uncatalog because DB2 does not check to see whether the DB exists, it  
blindly catalogs it.  
    #-- so that would cause the rest of the script to fail.  
    #--  
    db2 db2start  
    db2 catalog database $DatabaseName  
    db2 catalog database $DatabaseName  
    db2 drop database $DatabaseName
```

```

db2 uncatalog database $DatabaseName

echo "Create the Trade database"
db2 db2start
db2 create database $DatabaseName
db2 db2start
db2 force application all
db2 db2stop
db2 db2start
db2 connect to $DatabaseName user $db2userid using $db2password
db2 -tvf $ScriptDirectory/Table.ddl
db2 db2start
db2 disconnect $DatabaseName
db2 connect to $DatabaseName user $db2userid using $db2password
db2 db2start
db2 update db config for $DatabaseName using logfilsiz 1000
db2 update db cfg for $DatabaseName using maxappls 100
db2 db2stop
db2 db2start

#--
#-- cataloging is necessary to recover from a previous installation's left over
database
#-- please ignore any error messages emitted
#-- we also do an uncatalog because DB2 does not check to see whether the DB exists, it
blindly catalogs it.
#-- so that would cause the rest of the script to fail.
#--
# su - $db2userid -c "db2 catalog database $DatabaseName"
# su - $db2userid -c "db2 catalog database $DatabaseName"
# su - $db2userid -c "db2 drop database $DatabaseName"
# su - $db2userid -c "db2 uncatalog database $DatabaseName"

# echo "Create the Trade database"
# su - $db2userid -c "db2 db2start"
# su - $db2userid -c "db2 create database $DatabaseName"
# su - $db2userid -c "db2 connect to $DatabaseName user $db2userid using
$db2password"
# su - $db2userid -c "db2 -tvf $ScriptDirectory/Table.ddl"
# su - $db2userid -c "db2 disconnect $DatabaseName"
# su - $db2userid -c "db2 update db config for $DatabaseName using logfilsiz 1000"
# su - $db2userid -c "db2 update db cfg for $DatabaseName using maxappls 100"
# su - $db2userid -c "db2 db2stop"
# su - $db2userid -c "db2 db2start"

fi

exit 0

```

The Table.ddl response file

Example B-10 is a sample Trade.ddl file which was used for deploying the Trade6 application.

Example: B-10 Table.ddl

```
DROP TABLE HOLDINGEJB;
DROP TABLE ACCOUNTPROFILEEJB;
DROP TABLE QUOTEEJB;
DROP TABLE KEYGENEJB;
DROP TABLE ACCOUNTEJB;
DROP TABLE ORDEREJB;

CREATE TABLE HOLDINGEJB
(PURCHASEPRICE DECIMAL(10, 2),
HOLDINGID INTEGER NOT NULL,
QUANTITY DOUBLE NOT NULL,
PURCHASEDATE TIMESTAMP,
ACCOUNT_ACCOUNTID INTEGER,
QUOTE_SYMBOL VARCHAR(250));

ALTER TABLE HOLDINGEJB
ADD CONSTRAINT PK_HOLDINGEJB PRIMARY KEY (HOLDINGID);

CREATE TABLE ACCOUNTPROFILEEJB
(ADDRESS VARCHAR(250),
PASSWORD VARCHAR(250),
USERID VARCHAR(250) NOT NULL,
EMAIL VARCHAR(250),
CREDITCARD VARCHAR(250),
FULLNAME VARCHAR(250));

ALTER TABLE ACCOUNTPROFILEEJB
ADD CONSTRAINT PK_ACCOUNTPROFILE2 PRIMARY KEY (USERID);

CREATE TABLE QUOTEEJB
(LOW DECIMAL(10, 2),
OPEN1 DECIMAL(10, 2),
VOLUME DOUBLE NOT NULL,
PRICE DECIMAL(10, 2),
HIGH DECIMAL(10, 2),
COMPANYNAME VARCHAR(250),
SYMBOL VARCHAR(250) NOT NULL,
CHANGE1 DOUBLE NOT NULL);

ALTER TABLE QUOTEEJB
ADD CONSTRAINT PK_QUOTEEJB PRIMARY KEY (SYMBOL);

CREATE TABLE KEYGENEJB
(KEYVAL INTEGER NOT NULL,
KEYNAME VARCHAR(250) NOT NULL);

ALTER TABLE KEYGENEJB
ADD CONSTRAINT PK_KEYGENEJB PRIMARY KEY (KEYNAME);

CREATE TABLE ACCOUNTEJB
(CREATIONDATE TIMESTAMP,
OPENBALANCE DECIMAL(10, 2),
LOGOUTCOUNT INTEGER NOT NULL,
BALANCE DECIMAL(10, 2),
```

```

ACCOUNTID INTEGER NOT NULL,
LASTLOGIN TIMESTAMP,
LOGINCOUNT INTEGER NOT NULL,
PROFILE_USERID VARCHAR(250));

ALTER TABLE ACCOUNTEJB
ADD CONSTRAINT PK_ACCOUNTEJB PRIMARY KEY (ACCOUNTID);

CREATE TABLE ORDEREJB
(ORDERFEE DECIMAL(10, 2),
COMPLETIONDATE TIMESTAMP,
ORDERTYPE VARCHAR(250),
ORDERSTATUS VARCHAR(250),
PRICE DECIMAL(10, 2),
QUANTITY DOUBLE NOT NULL,
OPENDATE TIMESTAMP,
ORDERID INTEGER NOT NULL,
ACCOUNT_ACCOUNTID INTEGER,
QUOTE_SYMBOL VARCHAR(250),
HOLDING_HOLDINGID INTEGER);

ALTER TABLE ORDEREJB
ADD CONSTRAINT PK_ORDEREJB PRIMARY KEY (ORDERID);

ALTER TABLE HOLDINGEJB VOLATILE;
ALTER TABLE ACCOUNTPROFLEEJB VOLATILE;
ALTER TABLE QUOTEEJB VOLATILE;
ALTER TABLE KEYGENEJB VOLATILE;
ALTER TABLE ACCOUNTEJB VOLATILE;
ALTER TABLE ORDEREJB VOLATILE;

CREATE INDEX a.profile_userid on accountejb(profile_userid);
CREATE INDEX h.account_accountid on holdingejb(account_accountid);
CREATE INDEX o.account_accountid on orderejb(account_accountid);
CREATE INDEX o.holding_holdingid on orderejb(holding_holdingid);
CREATE INDEX o.orderstatus on orderejb(orderstatus);
CREATE INDEX o.ordertype on orderejb(ordertype);

```

The SetupProcs.jacl script file

Example B-11 is a sample SetupProcs.jacl file which was used for deploying the Trade6 application.

Example: B-11 SetupProcs.jacl file

```

# %I% %W% %G% %U%
#-----
# This script is used to set the values specified in
# the application response file
# It will set the variables required by the jacl scripts.
# Note that this may not include all of the
# variables specified in the response file.
#
# When adding a variable
# 1 - The variable must be defined as global
# 2 - set the variable value from the response file
# 3 - "puts" the variable and value

```

```

#      so it shows up in the log file
#-----

proc loadProperties {propFileName} {
    global env
    puts "SetupProcs - Loading properties"
    java::import java.io.FileInputStream
    java::import java.util.Properties

#-----
# Load the response file as Properties
#-----
    set props [java::new Properties]
    set fileStream [java::new FileInputStream $propFileName]
    $props load $fileStream

#-----
# The scripts directory is used to find the scripts included in the user program fileList
# as specified in the application.xml
#-----
    global scriptsDir
    set scriptsDir [string trim [$props getProperty TRADE.scriptsDir ]]
    puts " scriptsDir = $scriptsDir "

#-----
# Set the WAS configuration variables
#-----
    global wasServerName installableAppsDir appInstallFile contextRoot JDBCProviderName
    JDBCclasspath
    global dataSourceName datasourceDesc datasourceCategory
    global authAliasName authAliasDesc
    global appName

    global JMSauthAliasName JMSauthAliasDesc
    global DefaultOSUser DefaultOSPasswd

    set wasServerName [string trim [$props getProperty WAS.serverName ]]
    puts " wasServerName = $wasServerName "
    set installableAppsDir [string trim [$props getProperty WAS.installableAppsDir ]]
    puts " installableAppsDir = $installableAppsDir "
    set appInstallFile [string trim [$props getProperty WAS.appInstallFile ]]
    puts " appInstallFile = $appInstallFile "
    set JDBCProviderName [string trim [$props getProperty WAS.JDBCProviderName ]]
    puts " JDBCProviderName = $JDBCProviderName "
    set JDBCclasspath [string trim [$props getProperty WAS.JDBCclasspath ]]
    puts " JDBCclasspath = $JDBCclasspath "
    set dataSourceName [string trim [$props getProperty WAS.dataSourceName ]]
    puts " dataSourceName = $dataSourceName "
    set datasourceDesc [string trim [$props getProperty WAS.datasourceDesc ]]
    puts " datasourceDesc = $datasourceDesc "
    set datasourceCategory [string trim [$props getProperty WAS.datasourceCategory ]]
    puts " datasourceCategory = $datasourceCategory "
    set authAliasName [string trim [$props getProperty WAS.authAliasName ]]
    puts " authAliasName = $authAliasName "
    set authAliasDesc [string trim [$props getProperty WAS.authAliasDesc ]]
    puts " authAliasDesc = $authAliasDesc "
    set appName [string trim [$props getProperty WAS.appName ]]
    puts " appName = $appName "
    set contextRoot [string trim [$props getProperty WAS.contextRoot ]]
    puts " contextRoot = $contextRoot "

```



```

    set JMSauthAliasName [string trim [$props getProperty WAS.JMSauthAliasName ]]
    puts " JMSauthAliasName = $JMSauthAliasName "
    set JMSauthAliasDesc [string trim [$props getProperty WAS.JMSauthAliasDesc ]]
    puts " JMSauthAliasDesc = $JMSauthAliasDesc "

    set DefaultOSUser [string trim [$props getProperty DB2.UserID ]]
    puts " DefaultOSUser = $DefaultOSUser "
    set DefaultOSPasswd [string trim [$props getProperty DB2.UserPW ]]
    puts " DefaultOSPasswd = ***** "

#-----
# Set the DB2 variables
#-----
global databaseName db2instance db2UserId db2Password

    set databaseName [string trim [$props getProperty DB2.databaseName ]]
    puts " databaseName = $databaseName "
    set db2instance [string trim [$props getProperty DB2.db2instance ]]
    puts " db2instance = $db2instance "
    set db2UserId [string trim [$props getProperty DB2.UserID ]]
    puts " db2UserId = $db2UserId "
    set db2Password [string trim [$props getProperty DB2.UserPW ]]
    puts " db2Password = ***** "

    puts " "
    puts "End SetupProcs Script "
    puts " "
}

```

The Trade.prop script file

Example B-12 is a sample WebSphereScript.jacl file which was used for deploying the Trade6 application.

Example: B-12 Source of Trade.prop

```

#-----
#
# Trade Properties File
#
# HINTS:
#   - Set path names using shortname, e.g., "Program Files" is "progra~1"
#   - Use forward slash in directory entries
#
# Note that some of these values are set in Main
#-----

#   The name of the WebSphere configuration script to be run
TRADE.WASscript=WebSphereScript.jacl

#   The name of the DB2 script to be run
TRADE.DB2script=DB2Script.sh

TRADE.scriptsDir=Trade_ScriptsDir
# TRADE.scriptsDir=Trade_ScriptsDir

```

```

# The scripts are part of the userPrograms fileList in the application.xml.
# The scriptsDir is the directory under which the scripts were placed, as
# specified in the application.xml.
# TRADE.scriptsDir=Trade_ScriptsDir

# Variables used for WAS
WAS.version=6.0.0.0
WAS.serverName=server1
WAS.profile=default
# WAS installable apps directory is set in the Main
WAS.installableAppsDir=

# -----
# Variables used for WAS configuration steps
# -----

# The name of the application to be installed
WAS.appName=Trade

# the install EAR or WAR file
WAS.appInstallFile=trade.ear

# the contextRoot -- used if the application is installed using a war file
WAS.contextRoot=trade

# the JDBC classpath will be set in the Main --
c:/progra~1/IBM/SQLLIB/java/db2java.zip
WAS.JDBCclasspath=

# the JDBCProvider name to be created
WAS.JDBCProviderName=TradeDB2Provider

# the DataSource to be created
WAS.datasourceName=TradeDataSource
WAS.datasourceDesc="This datasource is used by the Trade Application"
WAS.datasourceCategory="Trade Application"

# the J2C Authorization entry to be created
WAS.authAliasName=TradeDataSourceAuthData
WAS.authAliasDesc="JAASDataAuth for the Trade database"

# the J2C Authorization entry to be created for JMS resource
WAS.JMSauthAliasName=TradeOSUserIDAuthData
WAS.JMSauthAliasDesc="JAASDataAuth for the Trade JMS resources"

# the Local OS userid and password
WAS.DefaultOSUser=LocalOSUserID
WAS.DefaultOSPasswd=password

# the DB2 database name and instance
DB2.databaseName=tradedb
DB2.db2instance=DB2
# the DB2 Administrator userid and password
DB2.AdminID=
DB2.AdminPW=

# The name of the IHS service, in quotes, to be started using the net command,
# e.g., net start "IBM HTTP Server 6.0"
# If a web server service has not been defined on the system, do not provide a value
# IHS.webServerService="IBM HTTP Server 6.0"

```

```
IHS.webServerName=webserver1
```

```
#      the DB2 database name and instance  
#DB2 version to check on the rpm database  
DB2.version=IBM_db2das81-8.1.0
```

```
#      WAS ISMP vpd.properties unique identifier (UID), used to determine WAS install dir  
WAS.vpdUID=WSBAA60
```

```
#      IHS vpd.properties unique identifier (UID), used to determine IHS install dir  
IHS.vpdUID=IHS6
```

```
#      the DB2 Administrator userid and password  
DB2.UserID=  
DB2.UserPW=
```

Archived

Source code for Flight400 user programs and script files on OS/400

This appendix provides the source codes for the Flight400 solution example. The following programs and files are included:

- ▶ application.xml
- ▶ solution.xml
- ▶ WebSphereScript.jacl
- ▶ SamplePDC.java
- ▶ SampleMain.java
- ▶ SampleExit.java
- ▶ SampleCommon.java
- ▶ SampleCommands.java

The application.xml file

Example C-1 shows the source of the application.xml file.

Example: C-1 Source of application.xml

```
<?xml version="1.0" ?>

<iru:application
  id="Flight400"
  xmlns:iru="http://www.ibm.com/xmlns/prod/iru/application"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/xmlns/prod/iru/application IRU_application.xsd">

  <applicationInformation
    installTime="12"
    version="1.0">
    <name>Flight400</name>
    <operatingSystems>
      <operatingSystem>OS/400</operatingSystem>
    </operatingSystems>
  </applicationInformation>

  <translationLanguages default="english">
    <language>english</language>
  </translationLanguages>

  <externalJars>
    <externalJar>../../externalSupportJars/jt400Native.jar</externalJar>
  </externalJars>

  <fileLists>
    <fileList id="softwareimagefiles">
      <file>Flight400.ear</file>
    </fileList>
    <fileList
      id="userprogramfiles"
      userPrograms="true">
      <file>com/ibm/flght400/SampleCommands.class</file>
      <file>com/ibm/flght400/SampleCommon.class</file>
      <file>com/ibm/flght400/SampleExit.class</file>
      <file>com/ibm/flght400/SampleMain.class</file>
      <file>com/ibm/flght400/SampleMessagesNLS.class</file>
      <file>com/ibm/flght400/SampleMessagesNLS_en.class</file>
      <file>com/ibm/flght400/SampleNLSkeys.class</file>
      <file>com/ibm/flght400/SamplePDC.class</file>
      <file>Flight400_Scripts/CheckAppInstall.jacl</file>
      <file>Flight400_Scripts/WebSphereScript.jacl</file>
    </fileList>
  </fileLists>

  <preDeploymentChecker
    logFile="SamplePDC.log"
    programName="com.ibm.flght400.SamplePDC"
    responseFile="Flight400.prop"
```

```

        successType="returnCode"
        type="java">
            <arguments>
                <argument responseFile="true" />
                <argument>V5R2</argument>
            </arguments>
        </preDeploymentChecker>

```

```

<mainProgram
    logFile="SampleMain.log"
    programName="com.ibm.flight400.SampleMain"
    responseFile="Flight400.prop"
    successType="returnCode"
    type="java">
        <arguments>
            <argument responseFile="true" />
        </arguments>
    </mainProgram>

```

```

<exitProgram
    programName="com.ibm.flight400.SampleExit"
    successType="returnCode"
    type="java" />

```

```

<variables>
    <stringVariable
        maxLength="100"
        minLength="1"
        name="appSvrName"
        required="true">
            <labelText>Application Server name</labelText>
            <propertiesAssociations>
                <propertiesAssociation keyword="appSvrName" />
            </propertiesAssociations>
            <cidFileAssociations>
                <cidFileAssociation
                    keyword="WAS.serverName"
                    responseFileName="Flight400.prop" />
            </cidFileAssociations>
            <defaultData>IRAppSvr</defaultData>
            <inputValidation>
                <invalid>
                    <prefixes>
                        <prefix>0</prefix>
                        <prefix>1</prefix>
                        <prefix>2</prefix>
                        <prefix>3</prefix>
                        <prefix>4</prefix>
                        <prefix>5</prefix>
                        <prefix>6</prefix>
                        <prefix>7</prefix>
                        <prefix>8</prefix>
                        <prefix>9</prefix>
                    </prefixes>
                    <characters>!@%*:/\#;?=<^|'+&apos;&quot;;(){}[]&amp;,&lt;&gt;&lt;/characters>
                </invalid>
            </inputValidation>

```

```

</stringVariable>
<!-- ***** -->
<!-- Specify a variable to expose the IHS Server name. -->
<!-- ***** -->
<stringVariable
  name="httpServerName"
  required="true">
    <labelText>HTTP Server Name</labelText>
    <helpText>Enter the name of your HTTP server</helpText>
    <propertiesAssociations>
      <propertiesAssociation keyword="httpServerName" />
    </propertiesAssociations>
    <defaultData>IRHTTP</defaultData>
    <inputValidation>
      <valid>
        <prefixes>
          <prefix ignoreCase="true">$</prefix>
          <prefix ignoreCase="true">#</prefix>
          <prefix ignoreCase="true">@</prefix>
          <prefix ignoreCase="true">A</prefix>
          <prefix ignoreCase="true">B</prefix>
          <prefix ignoreCase="true">C</prefix>
          <prefix ignoreCase="true">D</prefix>
          <prefix ignoreCase="true">E</prefix>
          <prefix ignoreCase="true">F</prefix>
          <prefix ignoreCase="true">G</prefix>
          <prefix ignoreCase="true">H</prefix>
          <prefix ignoreCase="true">I</prefix>
          <prefix ignoreCase="true">J</prefix>
          <prefix ignoreCase="true">K</prefix>
          <prefix ignoreCase="true">L</prefix>
          <prefix ignoreCase="true">M</prefix>
          <prefix ignoreCase="true">N</prefix>
          <prefix ignoreCase="true">O</prefix>
          <prefix ignoreCase="true">P</prefix>
          <prefix ignoreCase="true">Q</prefix>
          <prefix ignoreCase="true">R</prefix>
          <prefix ignoreCase="true">S</prefix>
          <prefix ignoreCase="true">T</prefix>
          <prefix ignoreCase="true">U</prefix>
          <prefix ignoreCase="true">V</prefix>
          <prefix ignoreCase="true">W</prefix>
          <prefix ignoreCase="true">X</prefix>
          <prefix ignoreCase="true">Y</prefix>
          <prefix ignoreCase="true">Z</prefix>
        </prefixes>
        <characters ignoreCase="true">/$#@_abcdefghijklmnopqrstuvwxyz0123456789</characters>
      </valid>
    </inputValidation>
    <cidFileAssociations>
      <cidFileAssociation
        keyword="IHS.webServerName"
        responseFileName="Flight400.prop" />
    </cidFileAssociations>
  </stringVariable>
</variables>

</iru:application>

```


The solution.xml file

Example C-2 shows the source of the solution.xml file.

Example: C-2 Source of solution.xml

```
<?xml version="1.0" ?>

<iru:solution
  id="Flight400Solution"
  version="1.0"
  xmlns:iru="http://www.ibm.com/xmlns/prod/iru/solution"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/xmlns/prod/iru/solution IRU_solution.xsd">

  <solutionInformation>
    <title>Flight400 Solution</title>
  </solutionInformation>

  <translationLanguages default="english">
    <language>english</language>
  </translationLanguages>

  <tasks>
    <installTask
      isOptional="true"
      operatingSystem="OS/400">
      <description>Installing WAS Express</description>
      <applications>
        <application fileName="IRU2_1WASExpress6_0I50S_os400.ser" />
        <application fileName="IRU2_1WASConfigI50S_os400.ser">
          <variables>
            <variable
              hidden="false"
              id="appSvrName"
              sharedAs="appSvrNameShared" />
            <variable
              hidden="true"
              id="HTTPPort"
              sharedAs="HTTPPortShared" />
          </variables>
        </application>
      </applications>
    </installTask>
    <installTask
      isOptional="true"
      operatingSystem="OS/400">
      <description>Configure HTTP Server V5R2</description>
      <applications>
        <application fileName="IRU2_1IHS520I50S_os400.ser" />
        <application fileName="IRU2_1IHSConfigI50S_os400.ser">
          <variables>
            <variable
              hidden="false"
              id="httpServerName"
              readonly="false"
              sharedAs="httpServerNameShared" />
          </variables>
        </application>
      </applications>
    </installTask>
  </tasks>
</iru:solution>
```

```

        <variable
            hidden="false"
            id="HTTPPort"
            sharedAs="HTTPPortShared" />
        <variable
            hidden="true"
            id="appSvrName"
            sharedAs="appSvrNameShared" />
    </variables>
</application>
</applications>
</installTask>
<installTask
    isOptional="true"
    operatingSystem="OS/400">
    <description>Configure HTTP Server V5R3</description>
    <applications>
        <application fileName="IRU2_1IHS530I50S_os400.ser" />
        <application fileName="IRU2_1IHSConfigI50S_os400.ser" />
    </applications>
</installTask>
<installTask
    isOptional="true"
    operatingSystem="OS/400">
    <description>Install Flight400 application</description>
    <applications>
        <application fileName="Flight400_os400.ser">
            <variables>
                <variable
                    hidden="false"
                    id="appSvrName"
                    sharedAs="appSvrNameShared" />
                <variable
                    hidden="false"
                    id="httpServerName"
                    sharedAs="httpServerNameShared" />
            </variables>
        </application>
    </applications>
</installTask>
</tasks>

<variables>
    <sharedVariable
        name="HTTPPortShared"
        required="true">
        <defaultData>80</defaultData>
        <inputValidation>
            <valid>
                <ranges>
                    <range>1 to 65535</range>
                </ranges>
            </valid>
        </inputValidation>
    </sharedVariable>
    <sharedVariable
        maxLength="10"
        minLength="1"
        name="appSvrNameShared"

```

```

required="true">
  <defaultData>IRAppSvr</defaultData>
  <inputValidation>
    <invalid>
      <prefixes>
        <prefix>0</prefix>
        <prefix>1</prefix>
        <prefix>2</prefix>
        <prefix>3</prefix>
        <prefix>4</prefix>
        <prefix>5</prefix>
        <prefix>6</prefix>
        <prefix>7</prefix>
        <prefix>8</prefix>
        <prefix>9</prefix>
      </prefixes>
      <characters>!@%*:/\#;?=\^|+&apos;&quot;;(){}[]&amp;,&lt;&gt;</characters>
    </invalid>
  </inputValidation>
</sharedVariable>
<sharedVariable
  maxLength="10"
  minLength="1"
  name="httpServerNameShared"
  required="true">
    <defaultData>IRHTTP</defaultData>
    <inputValidation>
      <valid>
        <prefixes>
          <prefix ignoreCase="true">$</prefix>
          <prefix ignoreCase="true">#</prefix>
          <prefix ignoreCase="true">@</prefix>
          <prefix ignoreCase="true">A</prefix>
          <prefix ignoreCase="true">B</prefix>
          <prefix ignoreCase="true">C</prefix>
          <prefix ignoreCase="true">D</prefix>
          <prefix ignoreCase="true">E</prefix>
          <prefix ignoreCase="true">F</prefix>
          <prefix ignoreCase="true">G</prefix>
          <prefix ignoreCase="true">H</prefix>
          <prefix ignoreCase="true">I</prefix>
          <prefix ignoreCase="true">J</prefix>
          <prefix ignoreCase="true">K</prefix>
          <prefix ignoreCase="true">L</prefix>
          <prefix ignoreCase="true">M</prefix>
          <prefix ignoreCase="true">N</prefix>
          <prefix ignoreCase="true">O</prefix>
          <prefix ignoreCase="true">P</prefix>
          <prefix ignoreCase="true">Q</prefix>
          <prefix ignoreCase="true">R</prefix>
          <prefix ignoreCase="true">S</prefix>
          <prefix ignoreCase="true">T</prefix>
          <prefix ignoreCase="true">U</prefix>
          <prefix ignoreCase="true">V</prefix>
          <prefix ignoreCase="true">W</prefix>
          <prefix ignoreCase="true">X</prefix>
          <prefix ignoreCase="true">Y</prefix>
          <prefix ignoreCase="true">Z</prefix>
        </prefixes>
        <characters ignoreCase="true">/$#@_ .abcdefghijklmnopqrstuvwxyz0123456789</characters>
      </valid>
    </inputValidation>
  </sharedVariable>

```

```

        </valid>
    </inputValidation>
</sharedVariable>
</variables>

```

```
</iru:solution>
```

The WebSphereScript.jacl file

Example C-3 shows the source of the WebSphereScript.jacl file.

Example: C-3 Source of WebSphereScript.jacl

```

#-----
# installApp - Install the specified application ear file if an
#               application with the same name does not exist.
#-----
# Input parameters:
# - appName      - application name
# - ear          - ear file
# - serverName   - server name
#
# Variables used in the script:
#
# - deployejb    - deploy ejb (true|false)
# - deployws     - deploy webservices (true|false)
# - defaultBindings - use default binding (true|false)
# - earMetaData  - use MetaData from ear (true|false)

global AdminControl
global AdminApp

set appName [lindex $argv 0]
set ear [lindex $argv 1]
set deployejb false
set deployws false
set defaultBindings true
set earMetaData true
set wasServerName [lindex $argv 2]

set baseNode [$AdminControl getNode]
puts "base Node = $baseNode"
set target [subst {$baseNode $wasServerName}]

puts ""
puts "Installing application {$appName}..."

# Check if the application already exists

set appList [$AdminApp list]
foreach item $appList {
    if {[string first $appName $item] >= 0} {
        set app $item
        break
    }
}

```

```

}

if {[info exists app]} {
    puts " Application Name:    ${appName}"
    puts " Ear file:           ${ear}"
    if {[llength $target] == 1} {
        puts " Target Cluster:    [lindex $target 0]"
    } else {
        puts " Target Node:         [lindex $target 0]"
        puts " Target Server:        [lindex $target 1]"
    }
    puts " Deploy EJB:             ${deployejb}"
    puts " Deploy WebServices:     ${deployws}"
    puts " Use default bindings:    ${defaultBindings}"
    puts " Use Ear MetaData:       ${earMetaData}"

    set parms "-appname $appName"
    if {$deployejb == "true"} {
        append parms "-deployejb"
    }
    if {$deployws == "true"} {
        append parms "-deployws"
    }
    if {$defaultBindings == "true"} {
        append parms "-usedefaultbindings"
    }
    if {$earMetaData == "true"} {
        append parms "-useMetaDataFromBinary yes"
    } else {
        append parms "-useMetaDataFromBinary no"
    }

    if {[llength $target] == 1} {
        append parms "-cluster [lindex $target 0]"
    } else {
        append parms "-node [lindex $target 0] -server [lindex $target 1]"
    }

    set parms1 [subst {$parms}]

    puts "Starting application install..."

    set app [$AdminApp install $ear $parms1]

    puts "Install completed successfully!"
} else {
    puts "${appName} already exists!"
}

$AdminConfig save

puts "Starting the application"
set appManager [$AdminControl queryNames type=ApplicationManager,process=$wasServerName,*]
$AdminControl invoke $appManager startApplication $appName

$AdminConfig save

```

The SamplePDC.java file

Example C-4 shows the source code for the SamplePDC class.

Example: C-4 Source of SamplePDC.java

```
/*
 * Created on Mar 10, 2005
 *
 * This sample program demonstrates how to create a wrapper.
 * Based on your knowledge of your application,
 * use this code as an example to develop a custom wrapper.
 */
package com.ibm.flight400;

import java.io.File;

import com.ibm.as400.access.AS400;
import com.ibm.as400.resource.RSoftwareResource;
import com.ibm.jsdt.support.SupportOS400Base;
import com.ibm.jsdt.support.SupportOS400Helper;

/**
 * This is a sample class, created as part of the Redbook example.
 */
public class SamplePDC extends SupportOS400Base {
    private String ivWASProfileDir = null;
    private String ivAppName = null;
    private String ivWASServerName = null;
    private String ivWasProfileName = null;
    private String ivHttpServerName = null;
    private String ivScriptsDir = null;
    private SupportOS400Helper ivHelper = null;
    private String ivDB2AdminUserId = null;
    private String ivBaseRelease = null;

    public SamplePDC(){
    }

    /**
     * Initialize the object.
     * Retrieve input parameter
     * parm#0 = The response file name.
     * parm#1 = The minimal release that the sample can be installed on.
     */
    public int init(String[] args)
    {
        setMainResources(SampleCommon.SAMPLE_MESSAGES);
        ivHelper = (SupportOS400Helper) getHelper();

        if (args.length != 2) {
            logMessage(getResourceString(SampleNLSkeys.BAD_NUMBER_PGM_ARGS, new String [] {"SamplePDC", "2",
Integer.toString(args.length)}}));
            return FAILURE;
        }
        setResponseFileName(args[0]);
        setBaseRelease(args[1]);

        return SUCCESS;
    }
}
```

```

}

/**
 * The sample application requires WAS and HTTP to be install,
 * and the WAS and HTTP servers instances to be configured. This PDC will
 * - check to see if WAS is installed. (version specified in the properties file)
 * - check to see if HTTP is installed.
 * - check to see if the HTTP server name is configured.
 * - check to see if the WAS profile is configured.
 * - check to see if the OS release is equal or later than the base release
 * - check to see if the application is installed
 */
private int check()
{
    logMessage ("We are in PDC.check method");
    // get the values set in the properties file, properties file is mandatory
    if (getResponseFileName() == null || getResponseFileName().trim().equals("")) {
        logMessage(getResourceString(SampleNLSKeys.NO_PROPERTIES_FILE));
        return FAILURE;
    } else {
        if (getProperties() == FAILURE) return FAILURE;
    }

    logMessage ("Checking the OS level");
    //check to make sure the release is at the correct level
    if (isCurReleaseEqualOrGreater(getBaseRelease()) == false) {
        logMessage(getResourceString(SampleNLSKeys.INVALID_OS_RELEASE, new
String[] {getBaseRelease()}));
        return FAILURE;
    }

    //check to make sure HTTP is installed
    logMessage ("Checking if HTTP installed");
    AS400 as400 = new AS400();
    if (SampleCommon.isProductInstall(as400, SampleCommon.HTTP_PRODUCT_ID,
RSoftwareResource.RELEASE_LEVEL_CURRENT,
RSoftwareResource.PRODUCT_OPTION_BASE) == false) {
        logMessage(getResourceString(SampleNLSKeys.HTTP_NOT_INSTALLED, new
String[] {SampleCommon.HTTP_PRODUCT_ID}));
        return FAILURE;
    }

    //check to make sure WAS is installed
    logMessage ("Checking if WAS installed");
    if (SampleCommon.isProductInstall(as400, SampleCommon.WAS_PRODUCT_ID,
RSoftwareResource.RELEASE_LEVEL_ONLY,
SampleCommon.WAS_PRODUCT_OPTION) == false) {
        logMessage(getResourceString(SampleNLSKeys.WAS_NOT_INSTALLED, new
String[] {SampleCommon.WAS_PRODUCT_ID}));
        return FAILURE;
    }

    //check to see if HTTP server is configured.
    logMessage ("Checking if HTTP configured");
    if (isHTTPServerConfigured() == false) {
        logMessage(getResourceString(SampleNLSKeys.HTTP_SERVER_NOT_CONFIG, new
String[] {getHttpServerName()}));
        return FAILURE;
    }
}

```

```

        //check to see if WAS server is configured.
        logMessage ("Checking if WAS configured");
        if (isWASServerConfigured() == false) {
            logMessage(getResourceString(SampleNLSkeys.WAS_SERVER_NOT_EXIST, new
String[]{getServerName()}));
            return FAILURE;
        }

/* Don't need this
    //check to see if DB2 Admin Id Exists
    logMessage ("Checking if DB2 user exist");
    setUsername (getDb2AdminUserId());
    if ( ivHelper.doesUserExist (this) == false) {
        logMessage(getResourceString(SampleNLSkeys.NO_USER_PROF, getDb2AdminUserId()));
        return FAILURE;
    }

    //The next if should be in Main program. Access to the script file is not available yet.
    //check to see if the application is installed.
    logMessage ("Checking if application installed");
    if (isApplicationInstalled() == true) {
        return PDC_EXISTS;
    }
*/
    return PDC_DOES_NOT_EXIST;
}
/**
 * Get values from the properties file
 */
private int getProperties()
{
    int rc = SUCCESS;
    // The properties file name was set in the constructor

    // Log the contents of the properties file
    ivHelper.logNewLine(this);
    setFileName(getResponseFileName());
    ivHelper.logAppendFile(this);
    ivHelper.logNewLine(this);

    // get the values specified in the properties file that are needed by this PDC

    setKey(SampleCommon.SERVER_NAME_KEY); // server name to use to determine the application path
    setServerName(ivHelper.getResponseFileValue(this));
    if (getServerName() == null || getServerName().equals("")) {
        logMessage(getResourceString(SampleNLSkeys.NO_PROPERTY, SampleCommon.SERVER_NAME_KEY));
        rc = FAILURE;
    }

    setKey(SampleCommon.WAS_PROFILE_KEY); // WAS profile name
    setWasProfileName(ivHelper.getResponseFileValue(this));
    if (getWasProfileName() == null || getWasProfileName().equals("")) {
        if (getServerName() == null || getServerName().equals("")) {
            logMessage(getResourceString(SampleNLSkeys.NO_PROPERTY, SampleCommon.WAS_PROFILE_KEY));
            rc = FAILURE;
        } else {
            setWasProfileName(getServerName());
        }
    }
}
determineWasProfileDir(getWasProfileName());

```



```

        setKey(SampleCommon.APP_NAME_KEY); // the name of the application to be installed, it is required
        setAppName(ivHelper.getResponseFileValue(this));
        if (getAppName() == null || getAppName().equals("")) {
            logMessage(getResourceString(SampleNLSKeys.NO_PROPERTY, SampleCommon.APP_NAME_KEY));
            rc = FAILURE;
        }
        setKey(SampleCommon.SCRIPTS_DIR_KEY); // the name of the scripts directory
        setScriptsDir(ivHelper.getResponseFileValue(this));
        if (getScriptsDir() == null || getScriptsDir().equals("")) {
            setScriptsDir(" ");
        } else {
            // Add a slash to the end of the path if one doesn't exist
            if (!getScriptsDir().endsWith(File.separator)) {
                setScriptsDir(getScriptsDir() + File.separator);
            }
        }

        setKey(SampleCommon.IHS_SERVER_NAME_KEY); // the name of the scripts directory
        setHttpServerName(ivHelper.getResponseFileValue(this));
        if (getHttpServerName() == null || getHttpServerName().equals("")) {
            logMessage(getResourceString(SampleNLSKeys.NO_PROPERTY, SampleCommon.IHS_SERVER_NAME_KEY));
            rc = FAILURE;
        }

        return rc;
    }

    /**Checks to see if the version and release of the current machine
     * are at or higher than the version and release passed in.
     * For example, if a release of V5R2M0 was passed, and the
     * current machine is V5R1M0, then a false would be returned.
     * If the release of the current machine is V5R2M0 or later,
     * a true is returned.
     * @param release The base release to check against. Expected format of VXXRXXM. For example V5R3M0.
     * @return boolean True if the current machines release is the same or higher than the base release,
    else false.
    */
    private boolean isCurReleaseEqualOrGreater(String release)
    {
        //get the current release level of the machine
        SupportOS400Helper os4Helper = getOS400Helper();
        String level = os4Helper.getOSLevel(this);

        //if could not determine the current machines release, return false
        if (level == null || release == null || release.equals("")) {
            return false;
        }

        int versionInt = Integer.parseInt(release.substring(1,2));
        int releaseInt = Integer.parseInt(release.substring(3,4));
        //check the current level with one passed in.
        if (os4Helper.getOSRelease(this) >= releaseInt &&
            os4Helper.getOSVersion(this) >= versionInt) {
            return true;
        }
        return false;
    }
}

```

```

/**
 * Log a message to the logger. Used to simplify the steps
 * needed to write a message to the log.
 * @param message - String Message to log.
 */
public void logMessage(String message)
{
    setMessage(message);
    ivHelper.log(this);
}

/**Determine if the application is already installed.
 * @return true if the application is already installed, else false is returned.
 */
private boolean isApplicationInstalled(){
    // Run the WebSphere Admin application (wsadmin), the
    // -f option indicates a jac1 script is the next arg
    // After the script name, the arg for the jac1 script is:
    // - the application name
    String checkCommand = getWasProfileDir() + "bin" + File.separator +
        "wsadmin -conntype NONE -f " + ivHelper.getUnpackedDir(this) +
        getScriptsDir() + SampleCommon.CHECK_INSTALL_SCRIPT + // Script to run
        " " + getAppName(); // the application name

    // invoke and log the command
    SampleCommands command = new SampleCommands(this, ivHelper);
    int rc = command.invokeCommand(checkCommand, false, false);
    if (rc == 0) {
        //check the output for the APP_EXIST string
        if (command.getCommandOutput().indexOf(SampleCommon.APP_EXISTS) >= 0) {
            return true;
        } else {
            return false;
        }
    } else {
        return false;
    }
}

/**
 * Check to see if the HTTP server is already configured.
 * A check is made to see if the instance file for the server
 * exists. If a instance file exists, then we consider
 * the server to be configured.
 * @return boolean True if the HTTP server is configure, else returns false.
 */
public boolean isHTTPServerConfigured()
{
    //check if the server instance file exists
    setFileName(SampleCommon.HTTP_INST_DIR + "/" + getHttpServerName() + ".MBR" );
    if (getOS400Helper().fileExists(this) == true) {
        //HTTP server does not exist
        return true;
    }
    return false;
}

/**Determines if the name of the WAS profile
 * is already configured for WAS.
 * @param Name of the WAS server to check
 * @return True if the server exists. False if the name is not configured.

```

```

    */
    public boolean isWASServerConfigured ()
    {
        /*Check the path where the profile would
        * be configured.
        */
        String wasDir = getWasProfileDir();
        if (wasDir == null || wasDir.equals("")) {
            return false;
        }
        File p = new File(getWasProfileDir());
        if ( p.isDirectory() && p.exists() )
            return true;
        else
            return false;
    }

    /**
    * Determine and set the WAS Profile path.
    * @param profileName
    */
    private void determineWasProfileDir(String profileName){
        setWasProfileDir(SampleCommon.determineWasInstanceDir(this, ivHelper, getWasProfileName()));
    }

    /**
    *
    * Getters and Setters
    */

    /**
    * Get the server name.
    */
    private String getServerName()
    {
        return ivWASServerName;
    }

    /**
    * Set the server name.
    */
    private void setServerName(String serverName)
    {
        if (serverName != null) {
            ivWASServerName = serverName.trim();
        } else {
            ivWASServerName = serverName;
        }
    }

    /**
    * Get the WAS profile name.
    */
    private String getWasProfileName()
    {
        return ivWasProfileName;
    }

    /**

```

```

* Set the WAS profile name.
*/
private void setWasProfileName(String profName)
{
    if (profName != null) {
        ivWasProfileName = profName.trim();
    } else {
        ivWasProfileName = profName;
    }
}

/**
 * Get the directory within userPrograms where the scripts are located
 */
private String getScriptsDir()
{
    return ivScriptsDir;
}

/**
 * Set the directory where the application documents are to be installed.
 */
private void setScriptsDir(String scriptsDir)
{
    if (scriptsDir != null) {
        ivScriptsDir = scriptsDir.trim();
    } else {
        ivScriptsDir = scriptsDir;
    }
}

/**
 * Get the the name of the HTTP server instance.
 */
private String getHttpServerName()
{
    return ivHttpServerName;
}

/**
 * Set the the name of the HTTP server instance.
 */
private void setHttpServerName(String httpServer)
{
    if (httpServer != null) {
        ivHttpServerName = httpServer.trim();
    } else {
        ivHttpServerName = httpServer;
    }
}

/**
 * Get the application name.
 */
private String getAppName()
{
    return ivAppName;
}

/**

```

```

    * Set the application name.
    */
private void setAppName(String appName)
{
    if (appName != null) {
        ivAppName = appName.trim();
    } else {
        ivAppName = appName;
    }
}

/**
 * Get the the name of the DB2Admin Userid.
 */
private String getDb2AdminUserId()
{
    return ivDB2AdminUserId;
}

/**
 * Set the the name of the DB2Admin Userid.
 */
private void setDb2AdminUserId(String userId)
{
    if (userId != null) {
        ivDB2AdminUserId = userId.trim();
    } else {
        ivDB2AdminUserId = userId;
    }
}

/**
 * Set the release that is the minimul release
 * that the sample can be installed on. So the sample
 * can be configured on this release and any greater.
 * @param release Release in the format of VxRxMx. For example V5R2M0.
 */
public void setBaseRelease(String release){
    if (release != null) {
        ivBaseRelease = release.trim();
    } else {
        ivBaseRelease = release;
    }
}

/**
 * Get the base release of the lowest i5/OS release
 * that the sample can be installed on.
 * @return Base release in the format of VxRxMx.
 */
public String getBaseRelease(){
    return ivBaseRelease;
}

/**
 * Get the WAS Profile path.
 */
public String getWasProfileDir(){
    return ivWASProfileDir;
}

/**
 * Set the WAS Profile path.

```

```

    * @param wasDir
    */
    public void setWasProfileDir(String wasDir){
        if (wasDir != null) {
            ivWASProfileDir = wasDir.trim();
        } else {
            ivWASProfileDir = wasDir;
        }
    }
}
/**
 * ***** Main Routine *****
 * @param args
 * 1 - properties file
 * 2 - The minimul release that the sample can be installed on.
 */
public static void main(String args[])
{
    SamplePDC checker = new SamplePDC();
    int rc = checker.init(args);
    if (rc == SUCCESS) {
        System.exit(checker.check());
    }
    System.exit(rc);
}
}

```

The SampleMain.java file

Example C-5 shows the source code of the SampleMain class.

Example: C-5 Source of SampleMain

```

/*
 * Created on Mar 10, 2005
 *
 * This sample program demonstrates how to create a wrapper.
 * Based on your knowledge of your application,
 * use this code as an example to develop a custom wrapper.
 */
package com.ibm.flight400;

import java.io.File;

import com.ibm.jsdt.support.SupportOS400Base;
import com.ibm.jsdt.support.SupportOS400Helper;

/**
 * This is a sample class, created as part of the Redbook example.
 */
public class SampleMain extends SupportOS400Base {

    private SupportOS400Helper ivHelper = null;

    private String ivWasProfileName = null;

    private String ivWasProfileBinDir = null;

```

```

private String ivInstallableAppsDir = null;

private String ivWasServerName = null;

private String ivWasInstallDir = null;

private String ivWasScript = null;

private String ivScriptsDir = null;

private String ivAppName = null;

private String ivAppFile = null;

private String ivIHSServerName = null;

//helper class to run I50S commands
private SampleCommands ivCommand = null;

private static final String TEMP_RESPONSEFILE_NAME = "Flight400.prop";

public SampleMain() {
}

/**
 * Initialize the object Retrieve input parameters parm#0 = response file
 * name
 */
public int init(String[] args) {
    ivHelper = (SupportOS400Helper) getHelper();
    setMainResources(SampleCommon.SAMPLE_MESSAGES);
    ivCommand = new SampleCommands(this, ivHelper);
    if (args.length != 1) {
        logMessage(getResourceString(SampleNLSkeys.BAD_NUMBER_PGM_ARGS,
            new String[] { "SampleMain", "2",
                Integer.toString(args.length) }));
        return FAILURE;
    }

    // The properties file is in the same format as a response file
    setResponseFileName(args[0]);

    return SUCCESS;
}

/**
 * Install and configure the application This will - get the DB2 Admin
 * userid and password entered by the user (from the support framework
 * properties file) - get the values for other required variables from the
 * application properties file - get the WAS directories, both the install
 * dir and the bin dir - copy the EAR or WAR file to the WAS installableApps
 * directory - copy the sample documents to the specified directory - run
 * the DB2 script to create and populate the database - run the WAS script
 * to do the WAS configuration - generate the HTTP plug-in - restart the
 * HTTP Server
 */
private int install() {
    int rc = SUCCESS;

```

```

// get the values set in the properties file, properties file is
// mandatory
if (getResponseFileName() == null
    || getResponseFileName().trim().equals("")) {
    logMessage(getResourceString(SampleNLSKeys.NO_PROPERTIES_FILE));
    rc = FAILURE;
} else {
    rc = getProperties();
}

// Determine if WAS is installed and get the WAS profile directories
logMessage("Determining WAS dir");
if (rc == SUCCESS) {
    rc = determineWasDirs();
}

// Set values in the properties file
logMessage("Setting properties");
if (rc == SUCCESS) {
    setProperties();
}

logMessage("Copy EAR file to installableApps dir");
if (rc == SUCCESS) {
    // copy the application EAR or WAR file into the "installable apps"
    // directory
    rc = copyApp();
}

logMessage("Running installation script");
if (rc == SUCCESS) {
    // Run the WAS Script
    rc = runWebSphereScript();
    //If application is already installed, we treat this as success
    if (rc == SampleCommon.SKIP_APP_INSTALL) {
        rc = SUCCESS;
    }

    if (rc != SUCCESS){
        return rc;
    }
}

// Generate the HTTP Plugin
logMessage("Generating plugin");
if (rc == SUCCESS) {
    rc = generateHTTPPlugin();
}

// Restart the HTTP Server so the plugin will be enabled
logMessage("Restarting the HTTP server");
if (rc == SUCCESS) {
    rc = restartHTTPServer();
}

//If application is already installed, we treat this as success
if (rc == SampleCommon.SKIP_APP_INSTALL) {
    rc = SUCCESS;
}

```



```

        return rc;
    }

/**
 * Get values from the properties files
 */
private int getProperties() {
    int rc = SUCCESS;

    //Get parameters from the response file
    setKey(SampleCommon.SERVER_NAME_KEY); // server name to use for the
    // start command
    setServerName(ivHelper.getResponseFileValue(this));
    if (getServerName() == null || getServerName().trim().equals("")) {
        logMessage(getResourceString(SampleNLSKeys.NO_PROPERTY,
            SampleCommon.SERVER_NAME_KEY));
        rc = FAILURE;
    }

    setKey(SampleCommon.WAS_PROFILE_KEY); // WAS profile name
    setWasProfileName(ivHelper.getResponseFileValue(this));
    if (getWasProfileName() == null
        || getWasProfileName().trim().equals("")) {
        if (getServerName() == null || getServerName().equals("")) {
            logMessage(getResourceString(SampleNLSKeys.NO_PROPERTY,
                SampleCommon.WAS_PROFILE_KEY));
            rc = FAILURE;
        } else {
            setWasProfileName(getServerName());
        }
    }

    setKey(SampleCommon.WAS_SCRIPT_KEY); // WAS script
    setWasScript(ivHelper.getResponseFileValue(this));

    setKey(SampleCommon.SCRIPTS_DIR_KEY); // the directory within
    // userPrograms where the scripts
    // are located
    setScriptsDir(ivHelper.getResponseFileValue(this));

    setKey(SampleCommon.APP_FILE_KEY); // the ear or war file to be
    // installed
    setAppFile(ivHelper.getResponseFileValue(this));

    setKey(SampleCommon.APP_NAME_KEY); // the name of the application to be
    // installed, it is required
    setAppName(ivHelper.getResponseFileValue(this));

    setKey(SampleCommon.IHS_SERVER_NAME_KEY); // the name of the IHS server
    setIhsServerName(ivHelper.getResponseFileValue(this));
    return rc;
}

/**
 * Set values in the application properties file that are needed by the jac1
 * scripts
 */
private void setProperties() {

    // set the WAS installable apps directory in the properties file

```

```

setKey(SampleCommon.WAS_INSTALLABLE_APPS_DIR_KEY);
setKeyValue(getInstallableAppsDir());
ivHelper.setResponseFileValue(this);

// include the contents of the updated properties file in the log
ivHelper.logNewLine(this);
setFileName(getResponseFileName());
ivHelper.logAppendFile(this);
ivHelper.logNewLine(this);
}

/**
 * Copies the EAR or WAR to the WAS installableApps directory Returns
 * SUCCESS or FAILURE
 */
private int copyApp() {
    int rc = FAILURE;

    // Copy the ear or war file to the WAS installableApps directory
    setSource(ivHelper.getUnpackedDir(this) + getAppFile());
    setTarget(getInstallableAppsDir());

    if (ivHelper.fileCopy(this)) {
        logMessage(getResourceString(SampleNLSkeys.COPYFILE_SUCCESS,
            getSource() + "," + getTarget()));
        rc = SUCCESS;
    } else {
        logMessage(getResourceString(SampleNLSkeys.COPYFILE_FAIL,
            getSource() + "," + getTarget()));
        rc = FAILURE;
    }

    return rc;
}

/**
 * Uses wsadmin to run the WebSphere jac1 scripts
 */
private int runWebSphereScript() {
    int rc = SUCCESS;

    //check to see if the application is installed.
    logMessage("Checking if application installed");
    if (isApplicationInstalled() == true) {
        logMessage("Application is already installed");
        return PDC_EXISTS;
    }

    //Make sure the WebSphere is started server before the install of the
    // application
    startWebSphereServer();
    // Get the WebSphere script name to be ran.

    String script = getWasScript();
    logMessage("Accessing the script and trying to run it: " + script);
    if ((script != null) && !script.trim().equals("")) {
        String unpackedDir = ivHelper.getUnpackedDir(this);

        //copy the properties file to the unpacked dir which
        //is open read to for other users. The values are read

```

```

//the jacl script via wsadmin. On i5/OS
//wsadmin runs under the QEJBSVR user id, so it needs
//read access to this file.
setSource(getResponseFileName());
File tempFile = new File(getResponseFileName());
String tempResponseFile = unpackedDir + tempFile.getName();
setTarget(unpackedDir);

if (ivHelper.fileCopy(this)) {
    rc = SUCCESS;
} else {
    logMessage(getResourceString(SampleNLSkeys.COPYFILE_FAIL,
        getSource() + "," + getTarget()));
    return FAILURE;
}
setChmodValue("+rx");
setFileName(tempResponseFile);
ivHelper.chmodFilePermissions(this);

// The directory where the scripts are located are in the
// unpackedDir. A subdirectory may have been specified for these in
// the
// application.xml.

String fullScriptsDir = unpackedDir + getScriptsDir()
    + File.separator;

// Run the WebSphere Admin application (wsadmin)
// The -f option indicates a jacl script is the next arg
// After the script name, the args for the jacl script are:
// - directory where the scripts are located
// - the fully qualified properties file name
// - DB2 User ID
// - DB2 User Id's password
String command = getWasProfileBinDir() + "wsadmin -profileName "
    + getWasProfileName() + " -f " + fullScriptsDir + script + // Main
    // jacl
    // script
    " " + getAppName() + // the app name
    " " + getAppFile() + // app file name
    " " + getServerName(); //WAS server name

logMessage("This is the wsadmin command: " + command);
rc = ivCommand.invokeCommand(command, command, false, false);

//Logging the command output
String cmdOutput = ivCommand.getCommandOutput();
logMessage(cmdOutput);

if (rc != 0) {
    setMessage(getResourceString(SampleNLSkeys.INSTALL_FAILED));
    ivHelper.log(this);
}
} else {
    logMessage(getResourceString(SampleNLSkeys.NO_WAS_SCRIPT));
    rc = FAILURE;
}

return rc;
}

```

```

/**
 * Determine the WAS profile bin directory and installable directory.
 */
private int determineWasDirs() {
    int rc = SUCCESS;

    setWasInstallDir(SampleCommon.determineWasInstanceDir(this, ivHelper,
        getWasProfileName()));
    if (getWasInstallDir() == null) {
        logMessage(getResourceString(SampleNLSkeys.WAS_NOT_INSTALLED));
        rc = FAILURE;
    } else {
        // Set the WAS profile bin dir
        setWasProfileBinDir(getWasInstallDir() + "bin/");
        setInstallableAppsDir(getWasInstallDir() + "installableApps/"); // set
        // the
        // installableApps
        // directory
    }
    return rc;
}

/**
 * restart the HTTP Server
 */
private int restartHTTPServer() {
    // use the net command to stop the server, log an error but continue if
    // an error occurs
    String command = "ENDTCPSVR SERVER(*HTTP) HTTPSVR( "
        + getIhsServerName() + ")";
    int rc = ivCommand.invokeCLCommand(command, false);
    if (rc != 0) {
        // log an error if not successful
        logMessage(getResourceString(SampleNLSkeys.STOP_HTTPSERVER_FAIL));
        ;
    }
    // use the net command to start the server
    command = "STRTCPSVR SERVER(*HTTP) HTTPSVR( " + getIhsServerName()
        + ")";
    rc = ivCommand.invokeCLCommand(command, false);
    if (rc != 0) {
        // log an error if not successful
        logMessage(getResourceString(SampleNLSkeys.START_HTTPSERVER_FAIL));
        return FAILURE;
    }
    return SUCCESS;
}

/**
 * Start WAS Server
 *
 * @return Return code from the startserver command
 */
private int startWebSphereServer() {
    String command = SampleCommon.WAS_PRODDATA_PATH + File.separator
        + "bin" + File.separator + "startServer " + getServerName()
        + " -profileName " + getServerName();
    logMessage("This is the startServer command: " + command);
    int rc = ivCommand.invokeCommand(command, true, true);
}

```

```

        if (rc != 0) {
            setMessage(getResourceString(SampleNLSkeys.START_WASEXPRESS_FAIL));
            ivHelper.log(this);
        }
        return rc;
    }

    /**
     * Generate the HTTP Plugin
     */
    private int generateHTTPPlugin() {
        int rc = SUCCESS;

        // Use the WAS GenPluginCfg to generate the HTTP plug-in config file
        String command = SampleCommon.WAS_PRODDATA_PATH + File.separator
            + "bin" + File.separator + "GenPluginCfg -profileName "
            + getServerName() + " -webserver.name " + getIhsServerName()
            + " -node.name " + getIhsServerName() + "_node";
        if (ivCommand.invokeCommand(command, true, false) != 0) {
            rc = FAILURE;
        }
        return rc;
    }

    /**
     * Log a message to the logger. Used to simplify the steps needed to write a
     * message to the log.
     *
     * @param message -
     *             String Message to log.
     */
    public void logMessage(String message) {
        setMessage(message);
        ivHelper.log(this);
    }

    /**
     * *****
     *
     * Standard getters and setters
     *
     * *****
     */

    /**
     * Get the name of the ear or war file to be installed.
     */
    private String getAppFile() {
        return ivAppFile;
    }

    /**
     * Set the name of the ear or war file to be installed.
     */
    private void setAppFile(String appName) {
        ivAppFile = appName;
    }

    /**
     * Get the name of the WebSphere script to be run.

```

```

    */
private String getWasScript() {
    return ivWasScript;
}

/**
 * Set the name of the WebSphere script to be run.
 */
private void setWasScript(String scriptName) {
    ivWasScript = scriptName;
}

/**
 * Get the directory within userPrograms where the scripts are located
 */
private String getScriptsDir() {
    return ivScriptsDir;
}

/**
 * Set the directory where the application documents are to be installed.
 */
private void setScriptsDir(String scriptsDir) {
    ivScriptsDir = scriptsDir;
}

/**
 * Get the server name.
 */
private String getServerName() {
    return ivWasServerName;
}

/**
 * Set the server name.
 */
private void setServerName(String serverName) {
    ivWasServerName = serverName;
}

/**
 * Get the WAS profile name.
 */
private String getWasProfileName() {
    return ivWasProfileName;
}

/**
 * Set the WAS profile name.
 */
private void setWasProfileName(String profName) {
    ivWasProfileName = profName;
}

/**
 * Get the WAS bin directory name.
 */
private String getWasProfileBinDir() {
    return ivWasProfileBinDir;
}

```

```

/**
 * Set the WAS bin directory name.
 */
private void setWasProfileBinDir(String binDir) {
    ivWasProfileBinDir = binDir;
}

/**
 * Get the WAS install directory name.
 */
private String getWasInstallDir() {
    return ivWasInstallDir;
}

/**
 * Set the WAS install directory name.
 */
private void setWasInstallDir(String installDir) {
    ivWasInstallDir = installDir;
}

/**
 * Get the WAS installableApps directory name.
 */
private String getInstallableAppsDir() {
    return ivInstallableAppsDir;
}

/**
 * Set the WAS installableApps directory name.
 */
private void setInstallableAppsDir(String installableAppsDir) {
    ivInstallableAppsDir = installableAppsDir;
}

/**
 * Get the IHS server name.
 */
private String getIhsServerName() {
    return ivIHSServerName;
}

/**
 * Set the IHS server name.
 */
private void setIhsServerName(String serverName) {
    ivIHSServerName = serverName;
}

/**
 * Get the application name.
 */
private String getAppName() {
    return ivAppName;
}

/**
 * Set the application name.
 */

```

```

private void setAppName(String appName) {
    ivAppName = appName;
}

/**
 * ***** Main Routine *****
 */
public static void main(String[] args) {
    SampleMain sampleMain = new SampleMain();
    int rc = sampleMain.init(args);
    if (rc == SUCCESS) {
        rc = sampleMain.install();
    }
    System.exit(rc);
}

/**
 * Determine if the application is already installed.
 *
 * @return true if the application is already installed, else false is
 *         returned.
 */
private boolean isApplicationInstalled() {
    // Run the WebSphere Admin application (wsadmin), the
    // -f option indicates a jac1 script is the next arg
    // After the script name, the arg for the jac1 script is:
    // - the application name

    //getting profile dir
    String profileDir = SampleCommon.determineWasInstanceDir(this,
        ivHelper, getWasProfileName());
    String checkCommand = profileDir + "bin" + File.separator
        + "wsadmin -conntype NONE -f " + ivHelper.getUnpackedDir(this)
        + getScriptsDir() + File.separator
        + SampleCommon.CHECK_INSTALL_SCRIPT + // Script
        // to
        // run
        " " + getAppName(); // the application name

    // invoke and log the command
    SampleCommands command = new SampleCommands(this, ivHelper);
    int rc = command.invokeCommand(checkCommand, false, false);
    if (rc == 0) {
        //check the output for the APP_EXIST string
        if (command.getCommandOutput().indexOf(SampleCommon.APP_EXISTS) >= 0) {
            return true;
        } else {
            return false;
        }
    } else {
        return false;
    }
}
}

```

The SampleExit.java file

Example C-6 shows the source code of the SampleExit class.

Example: C-6 Source of SampleExit

```
/*
 * Created on Mar 11, 2005
 *
 * This sample program demonstrates how to create a wrapper.
 * Based on your knowledge of your application,
 * use this code as an example to develop a custom wrapper.
 */
package com.ibm.flight400;

import com.ibm.jsdt.support.SupportOS400Base;
import com.ibm.jsdt.support.SupportOS400Helper;

/**
 * This is a sample class, created as part of the Redbook example.
 */

public class SampleExit extends SupportOS400Base {

    public SampleExit() {

    }

    /**
     * This method starts the WebFacing server on iSeries using the
     * invokeCLCommand method in SampleCommands class
     */

    private int startWFserver() {

        int rc = 0;
        SupportOS400Helper ivHelper = (SupportOS400Helper)getHelper();
        SampleCommands clHandler = new SampleCommands(this, ivHelper);
        String command = "STRTCPSVR SERVER(*WEBFACING)";

        //Invoke the command
        rc = clHandler.invokeCLCommand(command, true);

        if (rc != 0) {
            // log an error if not successful
            setMessage(getResourceString(SampleNLSkeys.START_WEBFACINGSERVER_FAIL));
            ivHelper.log(this);
        }

        return 0;
    }

    public static void main(String[] args) {
        SampleExit sampleExit = new SampleExit();
        int rc = sampleExit.startWFserver();
        System.exit(rc);
    }
}
```

The SampleCommon.java file

Example C-7 shows the source code of the SampleCommon class.

Example: C-7 Source of SampleCommon

```
/*
 * Created on Mar 10, 2005
 *
 * This sample program demonstrates how to create a wrapper.
 * Based on your knowledge of your application,
 * use this code as an example to develop a custom wrapper.
 */
package com.ibm.flight400;

import java.io.File;

import com.ibm.as400.access.AS400;
import com.ibm.as400.resource.RSoftwareResource;
import com.ibm.as400.resource.ResourceException;
import com.ibm.jsdt.support.SupportOS400Base;
import com.ibm.jsdt.support.SupportOS400Helper;

/**
 * This is a sample class, created as part of the Redbook example.
 */
public class SampleCommon {
    public static final String SAMPLE_FILES = "Flight400";
    public static final String CHECK_INSTALL_SCRIPT = "CheckAppInstall.jacl";
    public static final String APP_EXISTS = "APP_EXISTS"; // used in the checkInstall script
    public static final String APP_DOES_NOT_EXIST = "APP_DOES_NOT_EXIST"; // used in the checkInstall script

    //WAS install path
    public static String WAS_PRODDATA_PATH = "/QIBM/ProdData/WebSphere/AppServer/V6/Base";
    //Used in determining if the IHS server is configured
    public static final String HTTP_INST_DIR = "/QSYS.LIB/QUSRSYS.LIB/QATMHINSTC.FILE";
    //Used in determining if the correct products are installed
    public static final String HTTP_PRODUCT_ID = "5722DG1";
    public static final String WAS_PRODUCT_ID = "5733W60";
    public static final String WAS_PRODUCT_OPTION = "0001";
    public static final String SAMPLE_MESSAGES = "com.ibm.flight400.SampleMessagesNLS";

    public static final String[] WAS_WSADMIN_ERRORS = {
        "WASX7309W", // Configuration not saved
        "WASX7015E", // Exception occurred while running command
        "WASX7017E"}; // Exception received while running script

    // application response file keys
    public static final String SERVER_NAME_KEY = "WAS.serverName";
    public static final String WAS_SCRIPT_KEY = "Flight400.WASscript";
    public static final String WAS_PROFILE_KEY = "WAS.profile";
    public static final String DOCS_DIR_KEY = "Files.documentsDir";
    public static final String SCRIPTS_DIR_KEY = "Flight400.scriptsDir";
    public static final String WAS_INSTALLABLE_APPS_DIR_KEY = "WAS.installableAppsDir";
    public static final String APP_FILE_KEY = "WAS.appInstallFile";
    public static final String IHS_SERVER_NAME_KEY = "IHS.webServerName";
    public static final String APP_NAME_KEY = "WAS.appName";
    public static final String MODULELABEL = "Module:";
    public static final String URILABEL = "URI:";
```

```

public static final String SERVERLABEL ="Server: ";
public static final int SKIP_APP_INSTALL = 99;

/**
 * Get the WAS Install path as defined in the IFS file system
 * for WAS on i5/OS
 * @param wasVersion
 * @return the WAS install directory
 */
public static String determineWasInstanceDir(SupportOS400Base base, SupportOS400Helper helper, String
profileName)
{
    try {
        SampleCommands command = new SampleCommands(base, helper);
        //use the -getPath option on the wasProfile command to get the
        // path of the profile.
        String cmd = getWASBinDir() + "wasprofile -getPath -profileName " + profileName;
        if (command.invokeCommand(cmd, true, false) == 0) {
            String output = command.getCommandOutput().trim();
            if (output == null || output.equals("")){
                return null;
            }
            //grep out the first path in the output.
            //Assume it is a absolute path and starts with a /
            int startingIndex = output.indexOf(File.separator);
            int endIndex = output.indexOf(" ", startingIndex);
            String wasDir = null;
            if (endIndex > -1){
                wasDir = output.substring(startingIndex, endIndex);
            }else{
                wasDir = (output.substring(startingIndex));
            }
            //make sure it ends with a '/'
            if (!wasDir.endsWith("/")) {
                wasDir = wasDir + File.separator;
            }
            return wasDir;
        } else {
            return null;
        }
    } catch (Exception e) {
        base.setMessage(base.getResourceString(SampleNLSKeys.CMD_EXCEPTION, e.toString()));
        helper.log(base);
        return null;
    }
}

/**
 * Get the WAS install bin directory name.
 */
private static String getWASBinDir()
{
    return WAS_PRODDATA_PATH + File.separator + "bin" + File.separator;
}

/**
 * Checks the system to see if the product and option are install
 * on the machine for a specific version.

```

```

    * @param as400 - AS400 with a connection to the machine to check.
    * @param productId Product ID to check for. For example 5722DG1.
    * @param release i5/OS Release to check for. See RSoftwareResource Javadoc for constant values.
    * @param option Specific option to check for. See RSoftwareResource Javadoc for constant values.
    * @return True If the product, option is correctly installed for the release specified.
    *         False If the product is not installed, or if it is on the system, but not in a fully
installed state.
    */
    public static boolean isProductInstall( AS400 as400, String productId, String release, String option)
    {
        /*Initialize the RSoftwareResource with the ID, Option, Release that was
        * specified. Each RSoftwareResource represents one licenced product
        * on the machine.
        */
        RSoftwareResource product = new RSoftwareResource(as400, productId,
                                                         release, option);

        if (product == null) {
            return false;
        }
        try {
            //Get the load state of the product.
            String load_state = (String)product.getAttributeValue(RSoftwareResource.SYMBOLIC_LOAD_STATE);
            //a SYMBOLIC_LOAD_STATE_INSTALLED load state signifies that the product is successfully
loaded.
            if (load_state.equals(RSoftwareResource.SYMBOLIC_LOAD_STATE_INSTALLED)) {
                return true;
            } else {
                return false;
            }
        } catch (ResourceException e) {
            //No attributes, so the product is not loaded.
            return false;
        }
    }
}

/**
 * Determine if one or more messages exist in a string
 * @param source, the string to check
 * @param msgs, an array of message numbers
 * @return boolean, true if a message is found in the string
 */
public static boolean messageExists(String source,String [] msgs) {
    for (int i = 0; i < msgs.length; i++) {
        if (source.indexOf(msgs[i]) >= 0) {
            return true;
        }
    }
    return false;
}

/**
 * Helper method to parse the source string looking for sets of keywords(MODULE:,
 * URI:, and SERVER:) values. Source string may contain translated values for
 * the keywords Module: and Server:, this method replaces the translated values
 * with the English equivalent value.
 * @param source, the string to check
 * @return String, Module: <value> URI: <value> Server: <value> or null if none
 * found.
 */

```

```

public static String replaceTranslatedKeywords(String source)
{
    //This routine assumes the source string contains data returned from issuing the wsadmin
    //command of $AdminApp view DefaultApplication "-MapModulesToServers" . Further, the
    //contents of the source string is assumed to be in a specific order in the following format:
    //Module: value of module\n
    //URI: value of URI\n
    //Server: value of Server\n
    //The data returned from this call may contain translated keywords, the only assumption made
    //is the keyword of URI: is NOT translated. The other labels for Module: and Server: may be
    //translated and thus can not be used in the search algorithm. The search algorithm utilized
    //in this method is not the most elegant but it does provide a simple workaround for detecting
translated
    //keywords.
    String value = "";
    int moduleStart = 0, moduleEnd = 0, serverStart = 0, serverEnd = 0, totalLength = 0, previousLineLength
= 0, sourcePointer = 0;
    String[] lines = source.split("\n");

    for (int i = 0; i < lines.length; i++)
    {
        if (-1 != lines[i].indexOf(URILABEL))
        {
            if ((0 != i) && (i + 1 < lines.length))
            {
                moduleStart = totalLength;
                moduleEnd = moduleStart + lines[i-1].indexOf(":") + 1;
                serverStart = totalLength + lines[i-1].length() + 1 + lines[i].length() + 1;
                serverEnd = serverStart + lines[i+1].indexOf(":") + 1;
                if(0 != sourcePointer)
                {
                    value += source.substring(sourcePointer, moduleStart);
                }
                value += MODULELABEL;
                value += source.substring(moduleEnd, serverStart);
                value += SERVERLABEL;
                sourcePointer = serverEnd;
            }
        }
        totalLength += previousLineLength;
        previousLineLength = lines[i].length() + 1; // Add one for new line character
    }
    value += source.substring(sourcePointer, source.length());
    return (value);
}
}

```

The SampleCommands.java file

Example C-8 shows the source code of the SampleCommands class.

Example: C-8 Source of SampleCommands

```
/*
 * Created on Mar 10, 2005
 *
 * This sample program demonstrates how to create a wrapper.
 * Based on your knowledge of your application,
 * use this code as an example to develop a custom wrapper.
 */
package com.ibm.flight400;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

import com.ibm.as400.access.AS400;
import com.ibm.as400.access.CharConverter;
import com.ibm.as400.security.auth.AS400Credential;
import com.ibm.as400.security.auth.ProfileTokenCredential;
import com.ibm.jsdt.support.SupportOS400Base;
import com.ibm.jsdt.support.SupportOS400Helper;

/**
 * This is a sample class, created as part of the Redbook example.
 */
public class SampleCommands {
    private String ivCommandOutput = null;
    private SupportOS400Base ivBase = null;
    private SupportOS400Helper ivHelper = null;

    public SampleCommands(SupportOS400Base base, SupportOS400Helper helper){
        ivBase = base;
        ivHelper = helper;
    }

    /**
     * Run a QSH command.
     * To run a CL command use the invokeCLCommand() method.
     * The command and a success or failure message are logged.
     * The output can also be written to the log. The output
     * from the command can be gotten by calling getCommandOutput();
     * @param command the command to run.
     * @param boolean if the output of the command should be logged as well.
     * @param boolean if the output needs to be converted from EBCDIC to ASCII.
     * @return the return code of the command
     */
    public int invokeCommand(String command, boolean logOutput, boolean convertFromEBCDIC)
    {
        return invokeCommand(command, command, logOutput, convertFromEBCDIC);
    }

    /**
     * Run a QSH command.
     * To run a CL command use the invokeCLCommand() method.
     * The logCommand and a success or failure message are logged.
     */
}
```

```

* The output can also be written to the log. The output
* from the command can be gotten by calling getCommandOutput();
* @param command the command to run.
* @param command to write to the log - this is useful if the command contains
* sensitive information(e.g. passwords) that should not be written to the log.
* @param boolean if the output of the command to should be logged as well.
* @param boolean if the output needs to be converted from EBCDIC to ASCII.
* @return the return code of the command
*/
public int invokeCommand(String command, String logCommand, boolean logOutput, boolean
convertFromEBCDIC)
{
    int rc = -1;
    ivCommandOutput = "";
    try {
        // log the command
        ivHelper.logNewLine(ivBase);
        ivBase.setMessage(ivBase.getResourceString(SampleNLSKeys.CMD_INVOKED, logCommand));
        ivHelper.log(ivBase);

        Process p = Runtime.getRuntime().exec(command);
        rc = p.waitFor();

        // Log if the command passed or failed
        if (rc == 0) {
            ivBase.setMessage(ivBase.getResourceString(SampleNLSKeys.CMD_SUCCESS));
            ivHelper.log(ivBase);
        } else {
            ivBase.setMessage(ivBase.getResourceString(SampleNLSKeys.CMD_FAIL, new
String[] {String.valueOf(rc)}));
            ivHelper.log(ivBase);
            logOutput = true;
        }
        ivHelper.logNewLine(ivBase);
        ivCommandOutput = readCommandOutput( p.getInputStream(), convertFromEBCDIC);
        ivCommandOutput = ivCommandOutput + readCommandOutput(p.getErrorStream(), convertFromEBCDIC);
        if (logOutput == true) {
            ivBase.setMessage(ivCommandOutput);
            ivHelper.log(ivBase);
        }
    } catch (Exception e) {
        ivBase.setMessage(ivBase.getResourceString(SampleNLSKeys.CMD_EXCEPTION, new
String[] {e.getMessage()}));
        ivHelper.log(ivBase);
        rc = -1;
    }
    return rc;
}
/**
* Run a QSH command.
* To run a CL command use the invokeCLCommand() method.
* The command and a success or failure message are logged.
* The output can also be written to the log. The output
* from the command can be gotten by calling getCommandOutput();
* @param command the command to run.
* @param boolean if the output of the command to should be logged as well.
* @return the return code of the command
*/
public int invokeCommand(String command)
{

```

```

        return invokeCommand(command, false, false);
    }

    /**
     * Run a CL command and the write the command to the log.
     * To run a QSH command use one of the invokeCommand() method.
     * The output can be written to the log as well. The output
     * from the command can be gotten by calling getCommandOutput();
     * The output will not do any conversion of the data to ASCII.
     * @param command the command to run.
     * @param boolean if the output of the command to should be logged as well.
     * @return the return code of the command
     */
    public int invokeCLCommand(String command, boolean logOutput){
        ivCommandOutput = "";
        ivHelper.logNewLine(ivBase);
        ivBase.setMessage(ivBase.getResourceString(SampleNLSkeys.CMDINVOKED, command));
        ivHelper.log(ivBase);

        //set and call the CL command use the framework
        ivBase.setCommand(command);
        boolean brc = ivHelper.runCLCommand(ivBase);
        //save the output from the command
        ivCommandOutput = ivHelper.getLastCommandOutput(ivBase);

        int rc = 0;
        if (brc == true) {
            ivBase.setMessage(ivBase.getResourceString(SampleNLSkeys.CMD_SUCCESS));
            ivHelper.log(ivBase);
        } else {
            ivBase.setMessage(ivBase.getResourceString(SampleNLSkeys.CMD_FAIL, new
String[] {String.valueOf(rc)}));
            ivHelper.log(ivBase);
            logOutput = true;
            rc = 1;
        }
        if (logOutput == true) {
            ivBase.setMessage(ivCommandOutput);
            ivHelper.log(ivBase);
        }
        return rc;
    }

    /**
     * Run a QSH command under the user id specied.
     * Before running the command a swapuser is done to the new
     * user. Once the command is done running, another swapuser is done
     * to the original user.
     * The command is also written to the log and the
     * output can be written to the log as well. The output
     * from the command can be gotten by calling getCommandOutput();
     * @param swapUser the user to swap to before running the command
     * @parm swapPwd the password of the user
     * @param command the command to run.
     * @param boolean if the output of the command to should be logged as well.
     * @param boolean if the output needs to be converted from EBCDIC to ASCII.
     * @return the return code of the command
     */
    public int invokeCommandWithUser(String swapUser, String swapPwd, String command, boolean logOutput,
boolean convertOutputFromEBCDIC)

```



```

{
    int rc = 0;
    try {
        ivCommandOutput = "";
        //Swap to the new user. The current user's
        //credentials will be passed back. Keep them
        //to swap back later.
        AS400Credential credential = null;
        if (swapUser != null && !swapUser.equals("")) {
            credential = swapToUser (swapUser, swapPwd);
            //if there was an error swapping
            if (credential == null) {
                return 1;
            }
        }

        //Invoke the command under the new user profile
        invokeCommand(command, logOutput, convertOutputFromEBCDIC);
        //If we were able to swap users before the command
        //was ran, then swap back to the original user
        if (swapUser != null && !swapUser.equals("")) {
            rc = swapBack (credential, swapUser);
        }
    } catch (Exception e) {
        rc = 1;
        ivBase.setMessage(ivBase.getResourceString(SampleNLSKeys.CMD_EXCEPTION, e.toString()));
        ivHelper.log(ivBase);
    }

    return rc;
}

/**
 * Return the output from the last command ran.
 * @return String of the output from the last command ran.
 */
public String getCommandOutput(){
    return ivCommandOutput;
}

/**
 * Swaps to the a different user profile.
 * @param user String of the user to swap too
 * @param password String containing the password of the user.
 *
 * @return the credential object that can be used to swap back to user who called the
 *         method. If an error occurs <code>null</code> will be returned.
 */
private AS400Credential swapToUser (String user, String password)
{
    AS400 myMachine = new AS400 ();
    try {

        ProfileTokenCredential pt = new ProfileTokenCredential();
        pt.setSystem(myMachine);
        pt.setTimeoutInterval(60);
        pt.setTokenType(ProfileTokenCredential.TYPE_SINGLE_USE);
        // Note: The setTokenExtended() method was added to OS/400
        // V5R2 via PTF, at which time the setToken() method

```

```

// was deprecated. The technique below allows this code to
// successfully run regardless of whether or not the PTF making
// the change is present on the target system.
// Since the jt400Native.jar used during build must contain the new
// method, this will result in a deprecation warning for setToken().
try {
    pt.setTokenExtended(user, password);
} catch (NoSuchMethodError noMethodError) {
    // A compile time deprecation warning for the following
    // call is normal and expected.
    pt.setToken(user, password);
}

AS400Credential previousUserCredential = pt.swap (true);
//destroy credential for the DBUser
pt.destroy();

return previousUserCredential;

} catch (Exception e) {
    ivBase.setMessage(ivBase.getResourceString(SampleNLSkeys.SWAP_FAIL, user));
    ivHelper.log(ivBase);
    e.printStackTrace();
    return null;
}
}

/**
 * Attempts to swap to the given AS400Credential. The credential is destroyed after
 * the swap is performed.
 * @param userCredential AS400Credential of the user to swap to
 * @param currentUser String containing the userid of the current user
 * @return 0 on success 1 on failure.
 */
private int swapBack (AS400Credential userCredential, String currentUser)
{
    int rc = 0;
    try {
        userCredential.swap ();
        userCredential.destroy();
    } catch (Exception e) {
        ivBase.setMessage(ivBase.getResourceString(SampleNLSkeys.SWAP_FAIL, currentUser));
        ivHelper.log(ivBase);
        e.printStackTrace();
        rc = 1;
    }
    return rc;
}

/**
 * Reads the output from the commandOutput stream and converts it to ASCII
 * if requested. The output of the command is then returned.
 * @param commandOutput InputStream to read from.
 * @param convertFromEBCDIC boolean if to conversion from EBCDIC is needed.
 * @return The command output.
 */
private String readCommandOutput (InputStream commandOutput, boolean convertFromEBCDIC)
{
    try {

```

```

InputStreamReader inputStreamReader = null;
if (convertFromEBCDIC) {
    //get the "best guess" encoding based on the default locale
    String encoding = new CharConverter().getEncoding();
    inputStreamReader = new InputStreamReader (commandOutput, encoding);
} else {
    inputStreamReader = new InputStreamReader (commandOutput);
}
BufferedReader buf = new BufferedReader (inputStreamReader);
String line = buf.readLine ();
String buffer = "";
while (line != null) {
    if (buffer.length() == 0) {
        buffer = line;
    } else {
        buffer = buffer + "\n" + line;
    }
    line = buf.readLine ();
}
return buffer;
} catch (IOException e) {
}
return "";
}
}

```

Archived

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG246674>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG246674.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
Flight400Solution.zip	This is the project interchange file with the Flight400 application and solution projects.
Trade6LinuxSolution.zip	This is the project interchange file with the Trade6 on Linux on POWER application and solution projects.
Trade6WinSolution.zip	This is the project interchange file with the Trade6 on Windows application and solution projects.
Trade6WinSolutionMultisystem.zip	This is the project interchange file with the Trade6 on Windows application and solution projects that are designed for distributed deployment.

System requirements for downloading the Web material

For the hardware requirements for running the sample applications, see 4.2, “System prerequisites” on page 34.

How to use the Web material

Follow these instructions to set up the development environment for sample applications.

1. Install Express Runtime V2.1 on your workstation (see 4.4, “Installing the product” on page 37).
2. Create a subdirectory (folder) on your workstation, and download three ZIP files from the Web.
3. Start Express Runtime V2.1.
4. Select **File** → **Import** from the menu.
5. Select **Project Interchange** and click **Next**.
6. For the *From zip file* field, click **Browse**.
7. Navigate to the directory where you downloaded the three ZIP files.
8. Select one of the files and click **Open**.
9. You should see two (or more) names that correspond to the application and solution projects. Select all of them and click **Finish**.
10. Repeat these steps for the other two ZIP files.

You should have all projects imported into your workspace.

Trade6 solution for multiple systems

We include a more advanced version of the Trade6 solution for Windows as part of the downloads. This solution is designed to be deployed on multiple Windows systems. As a result, it includes more application projects than other sample solutions. These are the short descriptions of each project in this solution:

- ▶ *TradeWinSolution*: This solution allows for the deployment of the Trade6 application and middleware across multiple servers (Windows platform only). You can deploy WebSphere Application Server, DB2 and IBM HTTP Serve, and the console, each to a different server.

In addition to using the supplied application projects that install the middleware, this solution is built from several additional application projects for the deployment, configuration, and integration of Trade across each of the server machines. These application projects are described in the following points.
- ▶ *WebServerAddAdminId*: This application project adds an administration ID to the IBM HTTP Server. This ID is used by the WebSphere Application Server server that runs on a remote platform, when WebSphere Application Server connects to the IBM HTTP Server to propagate the Web server plug-in to IBM HTTP Server.
- ▶ *StartWebServerWin*: This application project ensures that the IBM HTTP Server is started after its deployment and configuration.
- ▶ *TradeDB2Win*: This application project performs the database only configuration required by the Trade application. It creates the tables that are used by Trade.
- ▶ *WebServerDefinition*: This application project defines the remote IBM HTTP Server to the WebSphere Application Server administration console. This allows WebSphere Application Server to be aware of the remote IBM HTTP Server and to propagate plug-in file updates to the remote server. It does this over a secured connection using the user ID

and password that were defined on the IBM HTTP Server in the application project WebServerAddAdminId.

- *TradeWASWin*: This application project deploys the Trade EAR file to the WebSphere Application Server. It completes all of the WebSphere Application Server configuration, including integration with the remote DB2 server, and propagation of the updated Web server plug-in file from WebSphere Application Server to the IBM HTTP Server.

Additional files to run the sample applications

You also need the EAR files for Trade6 and Flight400 applications. You can find the download instructions and other related information about the Flight400 application in the Additional Materials appendix in *Mastering the IBM WebFacing Tool*, SG24-6331. You can download the Flight400.ear file to any directory on your workstation.

You can download the Trade6 application from the following Web site:

<http://www.ibm.com/software/webservers/appserv/was/performance.html>

You see the Trade6 link under the Downloads section as shown in Figure D-1.

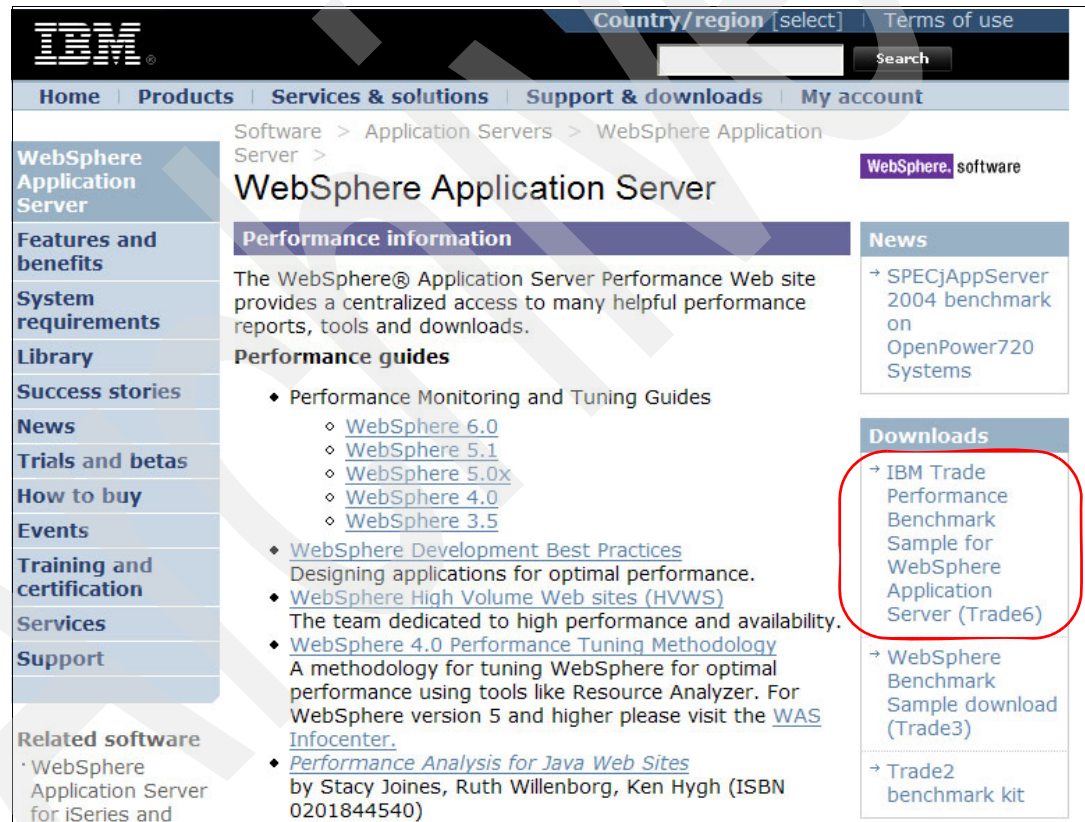


Figure D-1 Downloading Trade6

1. Download the ZIP file to any directory on your workstation.
2. Expand this file to a temporary directory.
3. You should see a new folder created - *tradeinstall*. Open this folder. You should see several files and directories. One of the files is *trade.ear*. You need this file for this book.

For our example, we place *trade.ear* in the C:\Trade folder. You need to create this folder on your PC.

Archived

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 439. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Rational Application Developer V6 Programming Guide*, SG24-6449
- ▶ *WebSphere Development Studio Client for iSeries Version 5.1.2*, SG24-6961-01

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ IBM HTTP Server page:
<http://www.ibm.com/software/webservers/htpservers/>
- ▶ WebSphere Application Server - Express page:
<http://www.ibm.com/software/webservers/appserv/express/>
- ▶ DB2 Universal Database Express Edition for Linux and Windows page:
<http://www.ibm.com/software/data/db2/udb/edition-express.html>
- ▶ IBM Express Runtime page at PartnerWorld®:
<http://www.developer.ibm.com/ir/expressruntime.html>
- ▶ Hardware and software prerequisites for WebSphere Application Server:
<http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html>
- ▶ DB2 Universal Database for Linux, UNIX®, and Windows support page:
<http://www.ibm.com/software/data/db2/udb/support/>
- ▶ DB2 UDB InfoCenter:
<http://publib.boulder.ibm.com/infocenter/db2help/index.jsp>
- ▶ WebSphere Application Server - Express V6.0 InfoCenter:
<http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

A

- A Target System 181
- Add a server to administer 241
- Add Application wizard 80
- administration console 6
- Appearance 84
 - Editable 84
 - Hidden 84
 - Read-only 84
- Applicable applications 6
- Application
 - Cheat Sheets 69
- Application Development Toolkit 20
- Application development tools 20
- application server model 4
- Application server tier 4
- Application wrapper 64
- Application Wrapper Editor 14, 68
- Application Wrapper in GUI
 - Cheat Sheets 69
 - Files 73
 - General Screen 70
 - Programs 71
 - Source screen 75
 - Variables 72
 - Welcome Screen 68
- application.xml 64, 67–68
- applications 7

B

- Backup databases 244
- Behavior 84
 - Application-defined 84
 - Modify the default value of 84
 - Share the value of variable name with other variables 84
- Benefits of Express Runtime V2.1 5
- BM Help facility 21
- Building the Trade6 application project 144
- Building the Trade6 solution project 146
- Business Framework
 - Web Architecture 4
 - Web Architecture with IBM products 5

C

- CheckAppInstall.jacl 92
- CheckAppInstall.sh 149
- CID response file association 107
- Client tier 4
- com.ibm.flight400 154
- Competitive pricing 5
- Compiling user programs 124
- complex scenarios 224

- Compressed product images 21
- Connection between IIA and target system 188
- consistent experience 6
- Console for Express Runtime 13, 25, 28, 233–234
- console window 144
- create the Predeployment user program class 115
- Creating installation CDs for a solution 169
- Creating the solution package for Trade6 166
- Custom program 64

D

- DB2 Control Center. 247
- DB2 script 64
- DB2 UDB Express 8.2 for Windows and Linux 247
- DB2 UDB system requirement 180
- DB2 Universal Database 4
- DB2/400 for OS/400 248
- DB2Script.bat 92
- DB2Script.sh 149
- Debugging user programs
 - required to debug a solution deployment 162
- Debugging userPrograms 162
- Deploying Console for Express Runtime 234
- Deploying from a staging server 182
- Deploying from CDs 207
- Deploying from CDs/DVDs 182
- Deploying to a local system 196
- Deploying to the remote systems 209
- deploying Trade6 using a staging server 168
- Deployment methods 181
- Deployment scenarios 195
- Deployment tasks 181
- Deployment Wizard 7, 15, 33, 181
- Deployment Wizard Only 37
- Deployment Wizard Only Installation 37, 46
- Deployment Wizard Only installation 195
- Deployment Wizard trace 232
- Developing Wrappers
 - Building the Trade6 solution 143
 - Completing the Trade6 application project 127
 - Creating the Trade6 application project 93
 - Creating Trade6 solution project 132
 - Creating Trade6 user programs 113
- Developing wrappers for Trade6 93
- development and deployment tools 13
- development environment 20
- development requirements
 - Linux development requirements 35
 - Linux on IBM POWER development requirements 36
 - Windows 35
 - Windows development requirements 35
- Development System 181
- diskIIA1 183
- diskIIA2 183

E

- EIS 4
- Enterprise Information Systems (EIS) tier 4
- Enterprise Java Beans 13
- Enterprise Java Beans (EJBs) 13
- enterprise level security 6
- Entry Program 67
- Entry program 67
- ER Plus 8
- ER V2.1 file directory 57
 - Linux & Linux on IBM Power 58
 - Windows 57
- Example of remote deploy of Integrated Solutions Console 234
- Exit program 67–68
- Exploring a wrapper 64
- Express Runtime 12
- Express Runtime (embed) 7
- Express Runtime (pre-req) 8
- Express Runtime Console 7
- Express Runtime Developer 7, 13
- Express Runtime First Steps 33
- Express Runtime Plus 9
- Express Runtime V2.1 Architecture 6

F

- FAILURE 115
- Firewall between Staging Server and Target Computer 189
- Flexible pricing and licensing 5
- Flight400 RPG application 152
- fully qualified host name 220

G

- Generating the solution 254
- getUnpackedDir 119

I

- i5/OS 35, 178
- i5/OS System Requirement 180
- IBM DB2 Universal Database Express Edition V8.2 21
- IBM Express Runtime Plus V2.1 9
- IBM Help System 33
- IBM HTTP Server 4
- IBM HTTP Server 6.0 21
- IBM HTTP Server 6.0 on OS/400 246
- IBM HTTP Server 6.0 on Windows and Linux 246
- IBM Installation Agent 25
- IBM Installation Agent (IIA) 181
- IBM Integrated Solutions Console 21
- IBM Rational Web Developer 7
- IBM Rational Web Developer V6.0 33
- IBM WebS Server plug-in 21
- IBM WebSphere Application Server - Express V6.0 21
- IBM WebSphere® Application Server - Express Version 6 247
- IIA 25
- IIA installation CDs directories structure 183

- IIA_LinuxPPCSetup 185
- IIA_LinuxSetup 185
- IIA_OS400Setup 190
- IIA_OS400Setup.exe 190
- IIA_WindowsSetup.exe 185
- Install IBM Express Runtime Components 37
- Install IBM Express Runtime Option 41
- Install only IBM Express Runtime Middleware Components 37
- Install only IBM Express Runtime middleware Components 37
- Install task 79
- Install tasks 80
- Installing IIA on i5/OS™ 190
- Installing on Windows®, Linux™ and Linux on POWER™ 184
- Installing the IBM Installation Agent (IIA) 183
- Installing the product 37
- InstallShield executable 64
- integrated administration 5
- Integrated Runtime 12
- Integrated Solution Console 25
- Integrated Solutions Console 234
- IRU_iaa_start-agent 193
- IRU2_1SampleI5OS 152
- IRU2_1SampleLnx 148
- ISC 25
- ISMP response file association 107
- ISS response file association 107

J

- JACL 64
- Java Builder 124
 - Building automatically 124
 - Building manually 124
 - trigger a build 124
- Java Messaging Services 13
- Java Messaging Services (JMS) 13
- Java program 64
- Jython 64

K

- Key Manger 185

L

- leavefiles 230
- Libraries 74
- Linux Intel system requirement 179
- Linux on POWER system requirement 179
- Local and remote deployment 182
- Log files 226
- Login to Console for Express Runtime 239

M

- Main Program 67
- Main program 68
- Managing Express Runtime 233
- Manual task 79

- Manual tasks 80
- Middleware
 - IBM® WebSphere® Application Server (WAS)- Express 6.0 6
- middleware 6
 - DB2 Universal Database Express Edition Version 8.2 12
 - IBM DB2 Universal Database™ Express Edition for Windows® and Linux™ 8.2 7
 - IBM HTTP Server 6.0 12
 - IBM HTTP Server 6.0 and HTTP plug-in 6
 - IBM HTTP Server 6.0 and IBM WebSphere HTTP plug-in 12
 - IBM WebSphere HTTP plug-in 12
 - WebSphere Application Server Express V6.0 12
- Middleware software 6
- Mid-market focus 5
- Migrating wrappers
 - Generating the solution 254
 - Moving projects and deployment package files 252
 - Updating wrapper files 253
- Migration
 - migrating an application that was included with Integrated Runtime V1.1 252
 - migrating projects from Integrated Runtime V1.1 to the Express Runtime V2.1 252
 - moving projects to another physical system 252
- Moving projects and deployment package files 252

O

- One single installation 6
- Open standards 6
- OS/400 35, 178
- Other ways for managing Express Runtime middleware 246
- Overridden application variables 80, 83
 - appearance 84
 - behavior 84

P

- Packaging a solution with just the application 175
- Passport 16
- Passport Advantage 8
- PDC_DOES_NOT_EXIST 115
- PDC_EXISTS 115
- Planning for the installation 34
- Platform Specific Instructions
 - Linux 38
 - Linux on IBM Power 39
 - Windows 38
- platform support 6, 16
- Predeployment Checker 67
- Predeployment checker 67
- Product Documentation 33
- Properties association 107
- properties file 110

Q

- QIIASRV 194

R

- Rapid return on investment 5
- Rational Web Developer 20
- Redbooks Web site 439
 - Contact us xi
- Reduce time to market 5
- response file 110
- Response file for silent installation of IIA 186
- restartHTTPServer() 122
- return code 67
- runDB2Script 120

S

- Sample wrappers and applications 33
- SampleExit.java 154
- SampleMain.java 154
- SampleMessageNLS.java 114
- SampleMessageNLS_xx(_xx).java 114
- SampleNLSKeys.java 114
- SamplePDC.java 154
- samples and documentation 6
- SampleWinMain.java 114
- SampleWinPDC.java 114
- Secure Key 196
- SetupProcs.jacl 92, 149
- setVariableName 120
- Silent Installation of IBM Installation Agent 186
- single install task 139
- Small to medium businesses 5
- SMB 5
- Solution Assembly Toolkit 33
- Solution Deployer 15
- solution project 77–78
- solution project wizard 77
- Solution wrapper 64
- Solution Wrapper Editor 77–78
- Solution Wrapper in GUI
 - General Screen 79
 - Source screen 86
 - Tasks Screen 80
 - Validation Screen 85
 - Welcome Screen 78
- solution.sxml 65, 77–78
- Source tab 75
- Staging Server 24, 181
- Start or stop a server 243
- Starting IBM Installation Agent on OS/400 193
- Support framework trace 232
- SupportBase 115
- Supported platforms 178
- SupportWindowsHelper 119
- System prerequisites 34
- System requirement 178

T

- Target Computer 183
- Target OS 34
- Task Group 139
- Task group 81
- Terminology used in this chapter 181
- The Main Program 97
- The Pre-deployment Checker 97
- The Trade.prop properties file 110
- The TradeWinMain.java program 118
 - tasks 118
- The TradeWinPDC.java program 115
- Tips for developing user programs 113
- Tracing on the target system 232
- Trade
 - Table.ddl 92
 - Trade.ear 92
- Trade.prop 120, 149
- Trade6 92
 - middleware application projects 143
 - The Main program 97
- Trade6 for Linux on POWER 148
- Trade6WinSolution.zip 93
- TradeLnxCommon 149
- TradeLnxMain 148
- TradeLnxPDC 148
- TradeMessagesNLS.java 149
- TradeMessagesNLS_en.java 149
- TradeNLSKeys.java 149
- tradewin.xx.jar 211
- tradewin.xx.userPrograms.jar 211
- TradeWinMain 113
- TradeWinPDC 113
- TradeWinSolution.ser 210
- Troubleshooting deployment 230
- Typical Installation 37, 43

U

- Uninstalling ER V2.1 58
- Updating wrapper files 253
- User programs 64
- user programs
 - developing 67
- Using Console for Express Runtime 239
- Using IBM Installation Agent 188

V

- Validating installation of IBM Installation Agent 188
- Validating installation on OS/400 193
- Validating the deployment 226
- valueIfFalse 108
- valueIfTrue 108
- Variable Validation Configuration 104, 107
- View log files 243

W

- WAS GenPluginCfg batch file 122
- Web server tier 4

- WebFacing 92
- WebFacing application 152
- WebFacing Tool 92
- WebSphere Application Server 4
- WebSphere Application Server system requirement 180
- WebSphere Portal Server 4
- WebSphereConfigProcs.jacl
 - 92
- WebSphereScript.jacl 92
- Welcome Screen 68
- Welcome Screen for Application Wrapper 68
- What if you have one components installed 224
 - Distributed platform 224
 - i5/OS™ platforms 226
- Windows 34, 178
- Windows system requirements 179



IBM Express Runtime V2.1

(0.5" spine)
0.475" <-> 0.873"
250 <-> 459 pages



IBM Express Runtime V2.1



The starting point for learning IBM Express Runtime V2.1

In-depth information on all supported platforms

Many useful examples

Mid-market companies are looking for complete solutions from the solution providers who know their businesses and specialize in a particular solution category such as warehouse management or supply chain integration, within their industry.

In order to provide an environment for the business solution, either the solution provider or the end customer must obtain the software needed to run the provider's application program. This software normally consists of a database, a Web server, and an application server. These components, which are not actually part of the business solution, but are required to support the application program, are called middleware.

The process of obtaining, installing, configuring, and supporting the middleware components is a complex process for the solution provider, who currently has to absorb the cost of setting up the environment. The provider has to work with multiple suppliers who have multiple terms and conditions, different license structures, and multiple departments to work with on the contracts and support structures, all of which leads to added infrastructure costs and inefficiency.

IBM designed the Express Runtime offering to address the needs of the solution providers serving mid-market customers. The main design criteria were reducing complexity for the providers by offering a pre-integrated and pre-configured middleware solution with a single contract, one license, one set of contracts, and a very competitive price.

With Express Runtime, you get WebSphere Application Server - Express, DB2 UDB Express, and IBM HTTP Server with a single install, one contract, and a significantly lower cost compared to purchasing the individual components. Express Runtime includes quick start samples and documentation to get you up and running quickly and easily.

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