JD Edwards EnterpriseOne Tools

Package Management Guide

9.2

JD Edwards EnterpriseOne Tools
Package Management Guide

9.2

Part Number: E53544-19

Contents

Preface

Welcome to the JD Edwards EnterpriseOne documentation.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Information

For additional information about JD Edwards EnterpriseOne applications, features, content, and training, visit the JD Edwards EnterpriseOne pages on the JD Edwards Resource Library located at:

`http://learnjde.com`

Conventions

The following text conventions are used in this document:

Convention	Meaning
Bold	Boldface type indicates graphical user interface elements associated with an action or terms defined in text or the glossary.
Italics	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`Monospace`	Monospace type indicates commands within a paragraph, URLs, code examples, text that appears on a screen, or text that you enter.
> Oracle by Example	Indicates a link to an Oracle by Example (OBE). OBEs provide hands-on, step- by-step instructions, including screen captures that guide you through a process using your own environment. Access to OBEs requires a valid Oracle account.

1 Introduction to JD Edwards EnterpriseOne Package Management

JD Edwards EnterpriseOne Package Management Overview

Oracle's JD Edwards EnterpriseOne Package Management describes how to set up and maintain processes to develop and deploy custom modifications that are created with Oracle's JD Edwards EnterpriseOne tools. You can use the guide to set up an environment in which you can deploy custom modifications that were made with development tools. It also provides information about applying modification rules, transferring objects, checking out development objects, and working with the data dictionary.

JD Edwards EnterpriseOne Package Management Implementation

JD Edwards EnterpriseOne standardizes and automates software installation, making many steps transparent to users. Technical setup is pre-configured to meet the requirements of many JD Edwards EnterpriseOne customers. In addition, JD Edwards EnterpriseOne products are pre-integrated and share a common database, which reduces the implementation process, minimizes ongoing administration, and provides customers with the flexibility to add new applications, modules, and tools as needed.

2 Understanding Package Management

Customer and Consultant Roles

Typically, both consultants and customers participate in an implementation. Consultants perform these roles:

- CNC consultant.
- Custom solution consultant.
- Application consultant.
- Hardware, network, and third-party software consultant.

Customers perform these roles, which parallel the consultant roles:

- CNC administrator.
- Application developer.
- Application project leader.
- Hardware, network, and third-party software administrator.

After the implementation, the consultants typically have fewer responsibilities. Therefore, customers must receive adequate training for the roles that they fill.

CNC Consultant and CNC Administrator

The CNC consultant and CNC administrator install JD Edwards EnterpriseOne and set up environments, users, security, and distributed processing. They also are responsible for setting up version control and testing various CNC configurations. The CNC consultant and CNC administrator control the deployment of JD Edwards EnterpriseOne software throughout the company.

Custom Solution Consultants and Application Developers

Custom solution consultants resolve business issues by developing applications. Their primary responsibilities include designing the modifications with upgrades in mind and developing, testing, and introducing the configured software. While the CNC administrator performs the version control functions that build and deploy software, the customer solution consultant must help develop the internal procedures that document the application development cycle for your business.

Application Consultants and Application Project Leaders

After JD Edwards EnterpriseOne software is installed, configured, and deployed, the application consultants continue in their role as product experts, where they might be called on to troubleshoot problems that arise. Although application consultants do not implement the CNC configurations, they must understand how JD Edwards EnterpriseOne handles distributed processing, environments, and so on, because these application issues influence the CNC decisions.

Hardware, Network, and Third-Party Software Consultants and Administrators

Implementing JD Edwards EnterpriseOne includes many tasks that are outside the scope of Oracle services. Third-party consultants provide these services. They might also work as CNC consultants, network architects, custom modification consultants, and so on.

Packages

The purpose of a package is to group software modifications so that they can be deployed to enterprise servers, web servers, and workstations. A package defines and contains the components to deploy. A package can contain everything that is needed to run the software (as in an initial installation for a new workstation), or only updates or changes to existing applications.

Why Packages Are Needed

As applications, business functions, and other objects change, you need to make those changes available to users within your enterprise. You might also need to set up a new workstation with JD Edwards EnterpriseOne software.

Packages enable you to deploy software changes and new applications to your users, or to install the software on a workstation for the first time. After you have defined and built a package, you can deploy it using Oracle's JD Edwards EnterpriseOne Client Workstation Installation application or Deployment Director application (P9631/P9631W).

You might need to update or set up a workstation or an enterprise, logic, or application server with JD Edwards EnterpriseOne software for a variety of reasons, such as:

- You want to set up a workstation for a new user or role.
- You need to deploy custom solutions to all users or only to selected users.
- You have created a new path code for development purposes, and you need to deploy it.
- You need to rapidly deploy a software fix to a selected group of affected users.
- Disk space is getting low on some of the workstations, and you need to create a minimum configuration.
- You need to update the servers with custom modifications that you have developed using the toolset.

JD Edwards EnterpriseOne provides solutions to meet all of these needs. First, the system enables you to create a package that defines and contains the location of the components that need to be distributed. To deploy these components, you must define and build a package.

Types of Packages

You can build these types of packages:

- Full client

- Full server
- Update client
- Update server

Full Client Packages

Full packages are static, point-in-time snapshots of the central objects for the path code on which the package is based. A full package contains everything that developers need to run in order to develop in JD Edwards EnterpriseOne software. Specifically, a full package includes a full set of specifications (specs), a full set of business function dlls, source files, object files, header files, bitmaps, an INF file that defines where the foundation and features are located, as well as other information about the package. Starting with Tools Release 9.2.5.0, the client package uses the package spec repository built under the database data source, Central Objects - `<pathcode>`.

In a full package, every application for which users are licensed is available to them. Starting with Tools Release 9.2.5.0, specification to run EnterpriseOne resides in the package tables located in Central Objects which eliminates the large footprint on the client machine.

Full packages are primarily for initial installations and are normally deployed using the JD Edwards EnterpriseOne Client Workstation Installation application. You can also use Oracle's JD Edwards EnterpriseOne Deployment Director application to install a full package on a computer on which JD Edwards EnterpriseOne is already installed.

Full Server Packages

Full server packages are the same as client packages except that:

- The server package does not include client-specific business functions.
- The server package does not include features.
- During the server package build process, the spec repository is built under the database data source, Central Objects - <pathcode>.

  This repository can be shared by several enterprise and web servers. The specs are built in a platform independent XML format. The spec repository is portable across dissimilar servers.
- Starting with Tools Release 9.2.5.0, the client package will now use the spec repository built under the database data source, Central Objects - <pathcode>.

Update Client Packages

The update package enables you to update, add to, or refresh the existing full package with changed objects. You can deploy an update package only to a workstation that already has JD Edwards EnterpriseOne installed on it. When the update package is deployed to a server, the objects are replaced in the Central Objects package database which enables the user to have access to those objects. When a user takes the update package and changes the objects in the update package, the system displays the application with the changed objects and asks if the user would like to delete the specs of the object. The user can delete the user specs and will be able to see the updates for that object. All other objects on the workstation are left unchanged. The advantage of this type of package is that users can quickly deploy software changes or enhancements.

When a user signs into JD Edwards EnterpriseOne and if an update package has been scheduled for that user, a list of available update packages appears. If the user decides to take one or more of these packages, the user spec application will appear if the user has user specs for the objects in the update package. The user can decide to delete the user specs. EnterpriseOne will then use the specs in central objects that was deployed or keep the user specs and then use the changes made by the user. All other objects on the workstation remain will the same.

Like full packages, update packages can include development objects such as business function source files, object specs, and header files. Update package recipients can load the development objects at deployment time.

All update packages require a full package on which the update package is based. This full package is called the parent package. While deploying to the server, the parent package is updated by the update package. When this occurs, all objects in the update package are merged into the parent package.

Business function objects in the update package are linked to the corresponding objects in the parent package, and new DLLs are created. Similarly, specs from the update package are merged into the specs in the parent package.

The parent package concept applies to both workstations and servers. Parent packages for workstations reside on the deployment server, while server parent packages are kept in the build area for the enterprise server.

Update Server Packages

Update server packages are the same as update client packages, with these exceptions:

- Update server packages include only the objects that were described previously for full server packages.
- When you are building an update server package, enterprise servers are automatically selected based on the parent package. Update packages can be deployed only to the enterprise and web servers that have the parent package deployed.
- Update packages must be parented to live full packages for them to be deployed.
- Specs are built in the database data source, Central Objects - <pathcode>.

Recommendations for Developers

When developers install a full package, they must select the option to install development objects. When they install the development objects with a full package, these objects will be automatically updated when developers later install an update package. Developers receive source code, header files, libraries, and DLLs.

Object Change Tracking

Managing modifications requires a practical version control plan for tracking changed objects. You can avoid many software problems by tracking changed objects.

To more easily plan and track development and to simplify version control, you should build and deploy packages only as often as necessary. If you perform many development changes, you should build and deploy packages on a set schedule to ensure that everyone involved knows when objects are due to be completed and when you are going to build and deploy the package.

Implementing version control might require a staff of information technology professionals. For example, a company has several hundred developers and a complex CNC configuration. To manage version control for multiple developers, the product version control group consists of:

- One manager to oversee coordination within the department.
- One supervisor to coordinate the package builds, coordinate object transfers, and troubleshoot problems.
- Two server specialists to build server packages.
- Four technical specialists to build workstation packages, perform object transfers, and run automated testing before releasing the package to Quality Assurance.
- One night operator to build workstation packages, build server packages, and clear build errors.

Path Codes

If you are not planning any development projects, you need only three path codes (sets of central objects): PY920, PD920, and PS920. If you plan to modify the software extensively, you also should create a development path code (DV920).

Because each path code requires version control maintenance, you should create only the path codes that you really need. Even when you make extensive software modifications, you should have only these four path codes:

DV920
The path code that you use for routine development. After successfully testing the objects that you develop, transfer them to the PY920 path code using the Object Transfer application, and distribute them to the users using the package build and deployment process.

PY920
A path code that contains a practice set of objects that you test during the conference room pilot before you transfer objects to production. Use this path code to deploy quick corrections or make minor modifications to objects that you will transfer to production. You also can use this path code to test modifications that were made in the development path code before you transfer the objects to the production path code.

PD920
The production path code from which just-in-time installations and production server objects are deployed. After you test software changes in PY920, transfer them to PD920, and then deploy the changes to the enterprise servers and workstations.

PS920
The set of pristine objects that is included with the software. You should not make changes to this path code other than to apply documented Oracle changes. Use this path code to compare standard software to any custom software that you have implemented in other path codes. You should keep a copy of this path code so that you have an unchanged copy of JD Edwards EnterpriseOne in case you need to undo any of the changes.

All path codes share the same Object Librarian tables, the same system data source, and usually, the same data dictionary. The only tables that are distinct for each path code are the central objects and specifications tables (tables that begin with F987), the F983051 table, the F98306 table, and the F98950 table.

Suggested Package Names

You should maintain two versions of each package, an A and a B version, so that you can alternate between these versions when you build packages. The advantage of this approach is that users always have a package available to them, even when you are building the latest version of that package. For example, package PRODB would be available to users while you are building PRODA. Then, after you release PRODA, you would build the next package into PRODB, and so on. This setup gives you two full packages (A and B) for production, as illustrated in this table:

Package Name	**Description**
PD920FA	Standard Production Full A
PD920FB	Standard Production Full B

Update packages might be named with the path code followed by "UA" or "UB" as shown:

Update Package Name	**Description**
PD920UA	Production Update Package 1
PD920UB	Production Update Package 2
PD920UA	Production Update Package 3
PD920UB	Production Update Package 4

A deployed server package cannot be revised, rebuilt, or deleted. It must be replaced with another package first. For this reason, you must have at least two server packages available so that you can alternate between them.

Note:

- *"Understanding Security Workbench" in the JD Edwards EnterpriseOne Tools Security Administration Guide* .

Note: The maximum length for a package name is nine characters unless the enterprise sever is IBM i. (Tools Release 9.2.7.3) If the enterprise server is IBM i, the addendum which is added to the package name library is taken into consideration and the maximum limit is 10 characters. For IBM i, if your package is DV920FA and the addendum is US, the package library will be DV920FAUS which is 9 characters and is valid. Any package with a package name and addendum greater than 10 characters will fail.

The Integrity of the Production Environment

As soon as you transfer objects into the PD920 path code, end users can access the changes. Therefore, you should test the modified objects before you transfer them to the production path code.

After you transfer objects to the production path code, they are immediately deployed to end users. When you build an update package that includes a business function build for that package, the system builds the business function and then globally links it with all other business functions in the parent package.

Do not transfer business functions into the production path code until you are ready to deploy because during a package build, a global build of business functions automatically includes the new business functions. When you transfer changes into the production path code, they will not be available to users until you build a full or update package.

The Normal Development Process

These lists provide an overview of how you should perform the normal development cycle.

In the DV920 path code, complete these tasks:

- Make modifications.
- Test the modifications.
- Transfer the objects to PY920.

In the PY920 path code, complete these tasks:

- Build the package.
- Deploy the server objects to the PY920 path code on the enterprise server, and then test the objects.
- Test the modifications.
- Schedule the package.
- Transfer the objects to PD920.

In the PD920 path code, complete these tasks:

- Build the package.
- Schedule the package.
- Deploy the server objects to the PD920 path code on the enterprise server, and then test the objects.

A Typical Development Process

The following steps are a typical process for modifying objects and deploying them through successive path codes and into the production environment.

1. Check out the objects from the DV920 path code, modify them, test them, and check them back in.
2. Use an incident-tracking system or any numbering system to track changes.

   Always use an incident number when you check in the objects.
3. If the objects need to reside on the logic server, transfer them to the DV920 path code on that server.
4. Test the objects by comparing them to the objects on the server.
5. Use an incident number with Object Transfer to transfer the objects to the PY920 path code.

   Use the checkout log to confirm the transfer (optional). The objects are not in production, but they are now available for you to build a test package in the PY920 path code.
6. Build a full or update package.
7. Test the newly built, unreleased package in the PY920 path code.

   You test the package only by comparing it to workstation processes, not to server processes. Although the name of this package will probably be PY920U1 (update package number 1 for the PY path code), it is a test package because you have not released it to the users.

8. Schedule the update package to deploy to a test machine and test it in an environment that contains PY objects with PY data.
9. Deploy server objects to the PY920 path code on the enterprise server and test them.

   If you prefer, you can build the server package and schedule the deployment at the same time that you build and schedule the workstation package. Building these packages simultaneously can save you time, although this method puts a greater load on the server.
10. Schedule the new package to deploy to PY users.
11. Use an incident number with Object Transfer to transfer the object to the PD920 path code.

    Use the checkout log to confirm the transfer (optional). The objects are now in the production environment and are available for you to build a package in the PD920 path code.
12. Build a full or update package for client workstations.
13. Perform a server package build.

    You can transfer the server package now or wait until it has been tested on a workstation.
14. Schedule the new package to deploy to end-user workstations.
15. Deploy the server objects to the PD920 path code on the enterprise server and test them.

    If you prefer, you can build the server package and schedule the deployment at the same time that you build and schedule the workstation package.

Developing Short-Term Changes

Sometimes you need to make a simple change to an application that is undergoing major enhancement work in the DV920 path code. When an object has been modified in the DV920 path code, you might get unexpected results if you make a simple change and quickly deploy it. Therefore, you should make the change in both the PY920 and the DV920 path codes, and deploy the change using the PY920 path code. This method enables you to deploy the change quickly to users without interfering with the major enhancement work in the DV920 path code.

Starting with Tools Release 9.2.8, the Promote feature in Web Object Workbench enables you to move the checked in objects from Central Objects to the package tables. The objects become available immediately to all users running the web OMW. See Promoting Objects in the Web Object Workbench Management Guide to learn more about promoting objects. This feature is the same as deploying of one or more objects for applications, UBE, business views, data structures, and media objects.

Deployment Methods

After you have made software changes, the method that you use to deploy those changes to the workstations on the enterprise depends on factors such as the type of package that you typically build and the needs of the users.

JD Edwards EnterpriseOne offers several deployment applications, each with its own specific purpose and advantages. The method that you select depends mainly on the type of package that you want to deploy.

This section discusses:

- Package deployment.
- Multitier deployment.
- Cumulative and noncumulative update packages.

- Comparing deployment methods.
- Deploying various types of modifications.
- Recommendations for sites using full packages with JITI.
- Disabling just-in-time installation.

Package Deployment

The JD Edwards EnterpriseOne Deployment Director application enables an administrator to deploy a built package to users, groups, locations, or enterprise servers. For clients, the administrator can specify the date and time when the package is made available and whether the package is mandatory or optional.

Users who receive a mandatory package will not be able to access JD Edwards EnterpriseOne until they install the package. Users who receive an optional package can install the package or decline it.

No option is available to schedule a server package for future deployment. The package is immediately deployed.

Multitier Deployment

Multitier deployment enables workstations to install software from more than one deployment location and more than one deployment server. You should consider multitier deployment if your site has more than 50 workstations performing software installations per day, or if you are deploying JD Edwards EnterpriseOne software across a WAN connection.

Cumulative and Noncumulative Update Packages

When you use a cumulative update strategy for deploying packages, you have one package that you add to, rebuild, and re-release to users. You do not create a new package each time you have a modification that you want to deploy. To use a cumulative package, follow these steps:

1. Change the package assembly status to **Inactive.**
2. Go to the Package Revisions form.
3. Add the changed or new objects to the package.
4. Rebuild the package.
5. Redeploy the package.

When you use a noncumulative update strategy, you create and deploy a different package each time you add or change an object. For example, if you deploy one modification a week for 10 weeks, you would have 10 different packages, each containing only the software change for that week.

Comparing Deployment Methods

Each deployment method has strengths and limitations. To help you decide which method is right for your needs, here are important points about the different methods:

- A new user loading a new machine should use the Install Manager to load a full package, plus any update packages that you have instructed users to load since the last package build.

Therefore, you need a manual tracking system to track which update packages must be applied after installing a particular package.

- All update packages must use the JD Edwards EnterpriseOne Deployment Director application to be scheduled to workstations.
- Full packages can also use JD Edwards EnterpriseOne Deployment Director if JD Edwards EnterpriseOne is already loaded on a machine.
- Use Oracle's JD Edwards EnterpriseOne Silent Installation application to submit a workstation installation request through command line arguments.

  Do not use this application for an initial installation.
- Use Oracle's JD Edwards EnterpriseOne Multitier Deployment process to install from more than one deployment location.

  You should consider this method if you have more than 50 workstations performing software installations per day or if you have users on a WAN.
- Use the JD Edwards EnterpriseOne Deployment Director application when you need to push objects in a server package to enterprise servers.
- Starting with Tools Release 9.2.8, use the Web OMW Promote function to deploy objects, applications, UBE, business view, data structure, and media objects from the Central Objects to the package tables, which is immediately available to the web users.

Deploying Various Types of Modifications

An understanding of which types of objects, and therefore modifications, can be deployed through each package type will help you select the appropriate package for the changes that you deploy.

This table shows which types of changes are installed with a full package, an update package with specs, and an update package with no specs.

Modification	**Full Package**	**Update Package with Specs, Business Functions, and Named Event Rules (NERs)**	**Update Package with Business Functions and NERs but no Specs**
Applications			
Imbedded event rules	X	X	
Vocabulary overrides (FDA text)	X	X	
Data structure	X	X	
Processing options (report)	X	X	

Modification	**Full Package**	**Update Package with Specs, Business Functions, and Named Event Rules (NERs)**	**Update Package with Business Functions and NERs but no Specs**
Business Functions			
C Language source/ include/object (if a compiler exists)	X	X	X
Consolidated business function DLLs	X	X	X
Data structure	X	X	
Table event rules	X	X	X
Named event rules	X	X	X
Batch Applications			
Report	X	X	
Imbedded event rules in a report	X	X	
Report data structure	X	X	
Report vocabulary overrides	X	X	
Report processing options	X	X	
Versions and processing option values (depends on processing options)	X	X	
Imbedded event rules in versions	X	X	
Processing option templates	X	X	
Business Views			
Added or changed fields	X	X	

Modification	**Full Package**	**Update Package with Specs, Business Functions, and Named Event Rules (NERs)**	**Update Package with Business Functions and NERs but no Specs**
Tables			
Structure (specifications)	X	X	
Indexes	X	X	
Joins	X	X	
Generic text data structure	X	X	
Data dictionary items			
Foundation code (required for full packages, optional for update packages)	X	X	X
Foreign languages	X	X	
Non-Oracle objects (custom items must be defined in the JD Edwards EnterpriseOne Central Objects database, and can be deployed through any package type)	X	X	
Replicated local data (required for full packages, optional for update packages)	X	X	X
New icons	X	X	

For an update package with JITI, these changes are installed:

- Applications.
- Imbedded ER.
- Vocabulary overrides (FDA text).
- Data structure.

- Processing options (report).
- Batch applications.
- Report.
- Imbedded ER in a report.
- Report data structure.
- Report vocabulary overrides.
- Report processing options.
- Versions and processing option values (depends on processing options).
- Imbedded ER in versions.
- Business Views.
- Added or changed fields.
- Miscellaneous.
- Foundation code (required for full packages, optional for update packages).
- Foreign languages.
- Non-Oracle objects (custom items can be deployed through any package type).

Package Implementation

This is an overview of the steps for creating and deploying a package:

1. Assemble the package.
   During this step, you specify the type of package that you are building and provide a name, path code, and package description. Next, you assemble the package by specifying the objects, foundation, features, and so on that you want to include in the package. If you are building an update package, you can specify individual objects to include.
   To simplify the process of assembling a package, Oracle's JD Edwards EnterpriseOne Package Assembly application (P9601/P9601W) includes the Package Assembly Director, which displays a series of forms that guide you through the steps of naming the package and adding the objects that you want to include in the package.
2. Define the package build.
   After you assemble the package, you must define the build before you can deploy the package to the workstations and servers. In this step, you specify:
   - Build options.
   - Build specification options.
   - Business functions build options.
   - Compression options.
   - Build features options.
     You also need to specify whether the package is for a workstation, a server, or both. If the package is for servers, you must specify the servers for which the package should be built and select the spec database data source.

To simplify the build process, Oracle's JD Edwards EnterpriseOne Package Build Director application (P9621/P9621W) includes the Package Build Definition Director, which displays a series of forms that guide you through the steps of specifying where to build the package, whether to include specifications, whether to compress or build business functions, and so on.

3. Build the package.

   During the actual build process, the system takes the information that you provided when you assembled and defined the package and copies and converts central objects to the package. It also globally builds the business functions that are included in the package and then compresses the package.

4. Schedule the package for deployment.

   After you have defined and built the package, it is ready for distribution. Depending on the package type, you can deploy packages through JD Edwards EnterpriseOne Client Workstation Installation application or Deployment Director application (P9631/P9631W).

   JD Edwards EnterpriseOne Deployment Director enables you to specify the workstations and servers that receive the package, as well as when the package is made available. Packages can be deployed to all computers within the enterprise, a select group of computers, or individual computers.

   When you schedule the package, you can indicate whether package installation is mandatory or optional.

5. Deploy the package to deployment, enterprise, and web servers.

   Use the JD Edwards EnterpriseOne Deployment Director application to move any changed objects to the enterprise server.

   If you specify a server during the package build definition process, the system automatically creates a corresponding server package in the correct format. If you do not specify a server and define only a workstation package, you should create a corresponding server package. The process is nearly identical to creating a workstation package.

   Web servers automatically retrieve the package information from their configured business function logic servers.

   See *Understanding Deployment to Web Servers*.

Package Implementation from the Web (Release 9.2.5)

Starting with Tools Release 9.2.5.0, you can assemble, define, and deploy a package from the web. A new deployment server service, JD Edwards B9 Client Network, which runs on the deployment server has been introduced. You control this service through the deployment server instance in the Server Manager and you can start and stop this service through the Server Manager. The JD Edwards B9 Client Network service accepts messages from the enterprise server and starts the client package build. Additionally, this service accepts the files that are being transferred from the enterprise server to the deployment server. This service must be running if you deploy a client package from the web.

To run a web package, a new service for the deployment server must be started during the Tool releases install of the server manager. Refer *"Change Component for the Deployment Server (Release 9.2.5)" in the JD Edwards EnterpriseOne Tools Server Manager Guide*

If you define a server or a client package and submit a build from the web when the JD Edwards B9 Client Network service is down, the package will fail because the service cannot connect to the deployment server to build the client package. However, if you define a server-only package and submit the build when the JD Edwards B9 Client Network service is down, the package will continue to run without connecting to the deployment server and the system will finish the package on the enterprise server.

When you run a server or a client package from the web, the port number of the enterprise server that the web is accessing must match the port number of the deployment server in jde.ini. To change the port number on the deployment server, access the Server Manager and select the deployment server instance. You must stop the services, change the port number in jde.ini, and restart the service to enable the web package to access the deployment server.

3 Understanding Objects

Objects

This section discusses:

- Object storage.
- Object movement.
- Performing backups and restoring objects.
- Correlating replicated and central objects.

Object Storage

By industry standards, an *object* is a self-sufficient entity that contains data as well as the structures and functions that are used to manipulate the data. An object is also a reusable entity that is based on software *specifications.* A specification is a complete description of a system object. Each object has one or more of its own specifications.

The system stores objects in four formats:

- Central objects are stored in a relational database format.

  Objects are stored in a central location to enable deployment and development. Central objects consist of object specifications for each JD Edwards EnterpriseOne object and C components for code-generated objects. Central object specifications are stored in a relational database on a data server.
- Package objects, or replicated objects, are stored in XML format in a relational database.

  Package objects are created during the package build process. You can specify the database data source in which to build the shared spec repository. This shared repository usually exists on a data server. Several enterprise servers and web servers can share this repository.
- Serialized objects are stored in database tables and used by web clients at runtime.

  Web servers use on-demand generation to create serialized objects from the shared spec data source. The generator converts specs into Java code, which enables you to access the JD Edwards EnterpriseOne applications in HTML. The system stores the objects in a relational database and retrieves them at runtime.
- File objects .c,.h,.hxx, and .jar, associated with business functions, tables, UBE, Business view and Business Services are stored in a Par file in the F98780R Repository table and F98780H Repository History table.

  Also, for every Application with a bit map, these are also stored in a Par file in this table. Every path code has one Central Objects table where these are stored. During package build these are retrieved from the F98780R and used in the package.

User Defined Objects

The following User Defined Objects (UDO) data is stored in the System Data Source:

- F9860W - Web Object Master Table
- F9861W - Web Object Librarian Detail

All other UDO data is stored in Central Objects tables.

Object Movement

When you perform any of these tasks, objects move between the central objects location and the object destination:

- Check in or check out objects.
- Add existing objects to a project.
- Perform a get object in Oracle's JD Edwards EnterpriseOne Object Management Workbench application (P98220).
- Run Oracle's JD Edwards EnterpriseOne Workstation Installation application.

  During the workstation installation, all objects and the Supported Local Database, which contains replicated data, are copied from the package to the workstation. The system copies objects in only those packages for which you included specifications. If the package does not include specifications, objects are not replicated; instead, they are installed on the workstation through just-in-time installation.

Unless otherwise noted, object movement is the same when you check objects in or out, add objects to a project, or perform a get object. This table describes the objects and specifications that move when you check in, check out, add, or get each type of object:

Object Type	Movement
Table (object type TBLE)	These objects move: • Table specs • Table event rule specifications (If the tables have event rules) • Source files (*.c) • Table header files (*.h) • Object files (*.obj) (If the objects have event rules) • Table event rule include files (*.hxx) (If the tables have event rules) The table header is not the same as the actual table that resides in a database. The table itself is created through Table Design Aid when you generate it. The JD Edwards EnterpriseOne Workstation Installation program copies this table to the workstation if the table is stored in the Supported Local Database.
Business view (object type BSVW)	These objects move: • Specifications • .h files (if generated)
C business function (object type BSFN, source language C)	These objects move:

Object Type	Movement
	• Specifications • Source files (*.c) • Header files (*.h) • Object files (*.obj)
NER business functions	Business function event rules (object type BSFN) can be checked out. When business function event rules are checked out, the .h file moves to the include directory, the .c file moves to the source directory, the .obj moves to the obj directory, and the local specifications are updated. These objects move: • Specifications • Source (*.c) • Header (*.h) • Object (*.obj)
Business function data structure (object type DSTR)	Specifications move.
Embedded event rules	You cannot check out embedded event rules. Embedded event rules move when you check out the object in which the event rule is embedded. For example, if embedded event rules are attached to a table, interactive application, or batch application, when you move the table or application, the specifications for the embedded event rules move with it.
Media object data structure (object type GT)	Data structure specifications move.
Interactive application (object type APPL)	These specifications move: • Application • Form • Form data structure • Embedded event rules
Batch application (object type UBE)	These objects move: • Report and event rule specifications (check in and check out the version separately from the report) • .h file (if generated by the developer)
Business Services (object type BSFN, source language Java) These objects are type BSFN except for: • F9860.SIPARDLL = " • F9860.SISRCLNG = "SBF"	These objects can be checked out and checked back in. The IDE for development of business services objects is JDeveloper. When they are checked out, the <F9860.SIOBNM>.zip file is copied from the path code check-in location on the deployment server (<path code>/java/sbfjars) to the <EnterpriseOne client install>/<path code>/java/source/<F9860.SIOBNM> folder on the Microsoft WIN32 client, and the contents are extracted. When they are checked in, the contents of the <EnterpriseOne client install>/<path code>/java/source/<F9860.SIOBNM> folder are compressed to a <F9860.SIOBNM>.zip file and the file is copied to the path code check-in location.

Object Type	Movement
User Defined Objects (such as Watchlists, queries, and grid formats) For a complete list, see the *JD Edwards EnterpriseOne Tools Using and Approving User Defined Objects Guide* .	User Defined Objects are web-based objects. They are managed with the Web Based Object Management application (P98220W). They can be administered via the Web Based Object Administration application (P9822U). These objects move differently than other objects. On check-in (share/publish), their components are consolidated into a single zip file and inserted as a single record to the F98700R (Object Archive Repository). This replaces the objects' existing repository record if one exists. The single record is also inserted into the F98700H (Object Archive History). This record will coexist with other history records for the same object. The UDO types may include the following data in .xml format: • One F9860W record. • One F9861W record. • Zero to many F9860WD records. • Zero to many F98700D records. • Zero or one F952460 record. Additional .xml data included in the .zip file for certain UDO types. F98700D.xml is created for the COMPOSITE UDO, which contains the object dependency information. Note that if there are no translation or dependency records, F9860WD.xml and F98700D.xml are not created. (Release 9.2.3) These objects are not included in traditional packages and do not require deployment to the servers. They are created on the servers and reside there immediately. On transfer, the object's current repository record is transferred from central objects in the source path code to central objects in the target path code (for example, DV920.F98700R to PD920.F9870R). After transfer, the User Defined Objects have a Pending Promote status in the target path code. These User Defined Objects need to be Approve/Shared in order to extract the data/artifacts from the F98700R repository to the runtime tables/locations. P98220W OMW Web by project or P98220U User Defined Object Administration by status can be used to perform the Approve/Share action. You can also use the Approve Promote option in the Work with Project Object Dependencies form to approve and share the objects, which are at the web object status of 07-Pending Promote in a project.

Note:

- *"JD Edwards EnterpriseOne OMW Overview" in the JD Edwards EnterpriseOne Tools Object Management Workbench Guide*

Performing Backups and Restoring Objects

You can back up development objects on workstations and servers as frequently as necessary.

Consider these scenarios and solutions when developing your backup strategy:

- A company does not allow the developers to back up directory data to the server because of space concerns. Developers are required to check in their development objects at specific time intervals, such as every eight hours, to avoid rework. Unless you have unlimited disk space on a file server to enable developers to back up their entire path code directory, you must use the check-in process as your backup method. If you follow the recommended development process, developers will know that they can check in unfinished or malfunctioning applications to the DEV path code.

- For workstation backups, end users should not have non-replicated data on their machines.
- For development server backups: At a certain company, the IT department backs up both the development file server (normally the deployment server) and necessary databases (central objects, Object Librarian, and data dictionary).

  When a developer needs to restore a particular object from backups, a database administrator restores the export to a path code called Restore. The developer checks out the object from Restore, ensures that the object functions as expected, and checks the object into the normal development path code.
- For deployment server backups: In most cases, you do not need to back up the entire server nightly.

  However, under certain conditions, you might need to back up these directories nightly:

  - The DEV path code, if you are modifying objects, building new packages, or updating the database that is delivered during a workstation installation.
  - MEDIA OBJ, if your media objects reside on the deployment server.
  - Data sources in Oracle or SQL Server, if your system data or any other important data is stored on the deployment server.
- For enterprise server backups:

  - Back up the DBMS nightly.

    You should use the backup tool that your database vendor provides.
  - Back up objects by backing up the entire directory.
  - Also back up the PROD and DEV path codes and the jde.ini file.

    Path codes are updated when the Version Control administrator deploys an object that was modified by developers who are authorized to access the Server Package application, and by end users who create new batch versions that will be run on the server.

Note:

- *"Backing Up JD Edwards EnterpriseOne Tables Servers" in the JD Edwards EnterpriseOne Administration Guide*
.

Correlating Replicated and Central Objects

This table describes the correlation between the spec data in the package build objects and the central objects that are stored in a relational database that has a binary large object (BLOB):

Replicated Package Build Object	Central Object
F98710<package name>	The F98710 table contains one record for each table.
F98711<package name>	The F98711 table contains one record for each column in a table.

Replicated Package Build Object	Central Object
F98712<package name>	The F98712 table contains one record for each table index.
F98713<package name>	The F98713 table contains one record for each column in an index.
F98720<package name>	The F98720 table contains one record for each business view.
F98740<package name>	The F98740 table contains one record for each event that has event rules for applications, reports, or tables. (Named event rule links are stored in the F9862).
F98741<package name>	The F98741 table contains one record for each line of event rules.
F98743<package name>	The F98743 table contains one record for each business function, processing option, form interconnection, report interconnection data structures, and power form.
F98750<package name>	The F98750 table contains override text for applications.
F98751<package name>	The F98751 table contains one record for every column, grid line, button, hyperitem, control, and so on, in the application.
F98752<package name>	The F98752 table contains one record for each application. If the application has processing options, that information is also stored in the record.
F98753<package name>	The F98753 table contains one record for each form (and also includes references to the data structures).
F98760<package name>	The F98760 table contains override text for batch reports.
F98761<package name>	The F98761 table contains one record for each section, column, sort, constant, and so on, in Batch Reports and Versions.
F98762<package name>	The F98762 table contains one record for each function in BSFN.
F98770<package name>	The F98770 table contains only one record for each package. This table is empty in Central Objects.
CGTYPE	Code Generator Form Types are stored in specification format only.
DDDICT DDTEXT	One record exists for each data dictionary item that has been just-in-time installed. This is data dictionary text.
GLBLTBL	This cache information from data dictionary and table specifications contains runtime table and override information. This is built dynamically the first time that a table is used.
SMRTTMPL	This is field information required by the data structure.

Replicated Package Build Object	Central Object
F9200 F9202 F9203 F9207 F9210	F9200 F9202 F9203 F9207 F9210
NEXTID	The F98701 table contains a local record of next IDs that are assigned to each workstation.

Modification Rules

This section discusses:

- Types of modifications.
- Objects that an upgrade preserves and replaces.

Types of Modifications

Because JD Edwards EnterpriseOne Development Tools are comprehensive and flexible, you can configure certain aspects of business solutions and applications without making custom modifications. This concept is referred to as *modless modifications.* Modless modifications are modifications that you can perform easily without the help of a developer. You can perform modless modifications on:

- User overrides
- User-defined codes (UDCs)
- Menu revisions
- All text
- Processing options values
- Data dictionary attributes
- Workflow processes
- User Defined Objects (UDOs)

This flexibility improves efficiency and provides distinct advantages, such as the ability to:

- Export grid records to other applications, such as a Microsoft Excel spreadsheet.
- Re-sequence a grid on a different column.
- Change grid fonts and colors.
- Control major system functions using processing options.

Developers may need to modify the JD Edwards EnterpriseOne software more extensively. To ensure that the modifications perform like modless modifications and to provide a seamless and predictable upgrade to the next release level, you should verify that any software modifications that you make comply with the recommended rules and standards.

To ensure a smooth upgrade, you should prepare for the upgrade before you make any custom modifications. If you plan modifications properly, you can minimize the tasks that you need to perform following an upgrade. Planning usually reduces the time that is required to upgrade your software, therefore reducing disruption to your business and the overhead cost of the upgrade.

The system tracks all custom modifications as you check them into the server. Before you perform an upgrade, you can run Oracle's JD Edwards EnterpriseOne Object Librarian Modifications report (R9840D) to see a list of the changed objects.

The system consists of control tables, such as menus, UDCs, versions, and the data dictionary, and transaction tables, such as the F0101 table. The system provides control tables, which contain data that you can modify, as well as transaction tables, which contain your business data.

During an upgrade, both sets of tables go through an automatic merge process. The system merges control tables with new data and converts transaction tables to the new specifications without changing your existing data. For the object specification merges (such as business views, tables, data structures, processing options, event rules, and applications), the system merges the specifications or replaces them, depending on the rules that are defined in the software.

Objects That a Traditional Upgrade Preserves and Replaces

Modification rules exist for these types of objects:

- Interactive applications
- Reports
- Application text changes
- Table specifications
- Control tables
- Business views
- Event rules
- Data structures
- Business functions
- Versions
- Business services
- User Defined Objects

General Rules for Modification

These general modification rules apply to all objects:

- When adding new objects, use system codes 55–59.

  The system uses its own reserved system codes that enable it to categorize different applications and vertical groups. When you use system codes 55 through 59 for your custom modifications, the system does not overlay your modifications with new applications.
- Do not create custom or new version names that begin with ZJDE or XJDE.

These prefixes are reserved for standard version templates that are included with the software, and these prefixes do not preserve your custom versions in case of a naming conflict. You can copy the pristine versions to create new templates or versions.

- For upgrades, build a package from the last modified central objects set and perform backups of your development server, central objects, and Object Librarian data sources so that you can access those specifications for comparison or for troubleshooting purposes.

Interactive Applications

Do not delete controls, grid columns, or hyper items on existing applications. If you do not want to see them, hide or disable them. The system might use these items for calculations or as variables, and deleting them might disable major system functions.

This table describes the interactive application elements that are preserved or replaced during an upgrade.

Object	Preserved	Replaced	Comments
New applications.	X		You can either create a new application, or copy an existing application using the Copy feature in Application Design Aid. This feature enables you to copy all of the application specifications, including event rules. If you use the Copy feature to copy an existing application for some modifications, during an upgrade your new application does not receive any changes that the system might have made to the original application.
New hyper items added to existing forms.		X	
New controls added to existing forms.		X	
New grid columns added to existing forms.		X	
Style changes.		X	Style changes include fonts and colors. New controls have the standard base definitions. If you adjust the style, you need to also adjust the styles for any new controls that you added to an application.
Code-generator overrides.		X	
Data dictionary overrides.		X	
Location and size changes.		X	In a subsequent release of the software, a new control might be placed in the same location that you have

Object	Preserved	Replaced	Comments
			placed a custom control. In this case, the new control appears on top of your custom control. This situation does not affect the event rules or the functions of the application. After the upgrade, you can use Application Design Aid to rearrange the controls.
Sequence changes for tabs or columns.		X	The upgrade process adds new controls to the end of your custom tab sequence. You can review the tab sequence after an upgrade.
Custom forms on existing applications.		X	Instead of adding custom forms to existing applications, create a custom application using system codes 55 through 59, and then place the custom form on that custom application. You can then add to existing applications Form exits and Row exits that call your custom forms within your custom applications. System performance is not adversely affected when you call an external application from a row exit instead of from a form within the application.

Note: None of the custom modifications to the JD Edwards EnterpriseOne applications are preserved during the Batch Specification Merge process. Instead, administrators must manually retrofit the modifications from a JD Edwards EnterpriseOne workstation with the help of Oracle's JD Edwards EnterpriseOne Visual ER Compare and FDA Compare tools when the upgrade is complete.

Reports

For Oracle's JD Edwards EnterpriseOne Report Design Aid specifications, do not delete objects on existing reports. Hide the objects that you do not want to appear. The system might use these objects for calculations or as variables, and deleting them could disable major system functions.

This table describes the report elements that are preserved or replaced during an upgrade:

Object	Preserved	Replaced	Comments
New reports.	X		You can either create a new report or copy an existing report using the Copy feature in Report Design Aid. This feature enables you to copy all the report specifications, including event rules. If you use the Copy feature to copy an existing report for some modifications, during an upgrade your new report does not receive any changes that might have been made to the original report.
New constants added to existing reports.	X		

Object	Preserved	Replaced	Comments
New alphabetical variables added to existing reports.	X		
New numeric variables added to existing reports.	X		
New data variables added to existing reports.	X		
New runtime variables added to existing reports.	X		
New database variables added to existing reports.	X		
New data dictionary variables added to existing reports.	X		
Style changes.	X		Style changes include fonts and colors. New controls have the standard base definitions. If you have adjusted the default style, you need to also adjust the styles for any new controls that you added to a report.
Location and size changes for objects.	X		In a subsequent release of the software, a new object, such as a control, might be placed in the same location as you placed a custom object. In this case, the objects appear next to each other. This situation does not affect the event rules or the functions of the report in any way. After the upgrade, you can use Report Design Aid to rearrange the objects.
Data dictionary overrides.	X		

Object	Preserved	Replaced	Comments
Custom sections on existing reports.		X	Instead of adding custom sections to existing reports, use Report Interconnect and connect to a new custom report that uses system codes 55 through 59. System performance is not adversely affected when you call a report through report interconnections.

Application and Report Text Changes

This table describes the application and report text that is preserved or replaced during an upgrade:

Object	Preserved	Replaced	Comments
Overrides performed in JD Edwards EnterpriseOne Application Design Aid.		X	Use the JD Edwards EnterpriseOne Visual Compare tools.
Overrides performed in JD Edwards EnterpriseOne Report Design Aid.	X		
Overrides performed in JD Edwards EnterpriseOne Interactive Vocabulary Override.		X	Use the JD Edwards EnterpriseOne Visual Compare tools.
Overrides performed in JD Edwards EnterpriseOne Batch Vocabulary Override.	X		

Table Specifications

An upgrade merges your table specifications from one release level to the next.

This table describes the table specification elements that are preserved or replaced during an upgrade:

Object	Preserved	Replaced	Comments
New Tables.	X		
Custom indexes to tables.	X		
Columns added to or removed from existing tables.		X	This object includes changing field length, field type, and decimal position. Instead of adding a new column to an existing table, use a tag table with system codes 55 through 59.

For custom tag files, be aware of data item changes in the data dictionary. In subsequent releases, JD Edwards EnterpriseOne software might contain changes to certain attributes of a data item, such as its size, that might affect data integrity and how the data is stored in the database. For this reason, you might need to use Oracle's JD Edwards EnterpriseOne Table Conversion tool to convert the tag file data to the new release level. For base files, the upgrade process automatically applies the data dictionary to the new release level. An upgrade preserves custom indexes for the custom tag files.

Control Tables

Control tables contain UDCs, menus, and data dictionary items. An upgrade merges your control tables from one release level to the next using the change table process, which uses your control tables, not system tables, as the basis for the data merge.

This table describes the control table elements that are preserved or replaced during an upgrade:

Object	Preserved	Replaced	Comments
Data dictionary custom changes.	X		This object includes changes to row, column, and glossary text. The upgrade process uses your data dictionary as the base, and in case of a conflict with system data items, your changes override. Create new data items using system codes 55 through 59.
UDCs.	X		The upgrade process merges any new, hard-coded values. (Oracle-owned values are stored in systems 90 and above, and H90 and above.) The process also reports any hard-coded values that conflict with your custom values.
Menus.	X		In case of a conflict with base menus, your custom changes override.
Columns added or removed from existing control tables.		X	

Business Views

Do not remove columns from existing business views. Changing business views that applications use can cause unexpected results when you run the application. If you need to hide columns, do so within the application design using either JD Edwards EnterpriseOne Application Design Aid or Report Design Aid. Deleting a few columns from a business view does not significantly improve system performance.

This table describes the business view elements that are preserved or replaced during an upgrade:

Object	Preserved	Replaced	Comments
New custom business views.	X		

Object	Preserved	Replaced	Comments
New joins that are added to existing business views.	X		
New columns or indexes that are added to existing business views.		X	
Columns that are removed from business views.		X	

Event Rules

The only JD Edwards EnterpriseOne objects that have event rules preserved are tables and UBEs. This table describes the event rule elements that are preserved or replaced during an upgrade:

Object	Preserved	Replaced	Comments
Custom event rules for custom applications, reports, and tables.	X		
Custom event rules for custom business functions.	X		
Custom event rules on a new custom control.		X	Use the JD Edwards EnterpriseOne Visual Compare tools.
Events for existing EnterpriseOne reports and tables that do not have any system event rules attached to the same event.	X		
Events for existing EnterpriseOne reports and tables that have existing event rules attached to the same event.		X	Use the JD Edwards EnterpriseOne Visual Compare tools.
Events for existing EnterpriseOne business functions or applications.		X	Use the JD Edwards EnterpriseOne Visual Compare tools.

To restore your custom event rules to system objects, highlight and drag the event rules back to the proper place in the event and enable them. Prior to an upgrade, perform these tasks:

- Run the JD Edwards EnterpriseOne Object Librarian Modifications report to identify modified applications.
- Print the event rules for the modified application so that you can see the logic for the event when you restore custom event rules.

Data Structures

This table describes the data structure elements that are preserved or replaced during an upgrade:

Object	Preserved	Replaced
Custom forms' data structures.	X	
Custom processing options' data structures.	X	
Custom reports' data structures.	X	
Custom business functions' data structures.	X	
Custom generic text's data structures.	X	
Modifications to existing EnterpriseOne forms' data structures.		X
Modifications to existing EnterpriseOne processing options' data structures.		X
Modifications to existing EnterpriseOne reports' data structures.		X
Modifications to existing EnterpriseOne business functions' data structures.		X
Modifications to existing EnterpriseOne generic text's data structures.		X

To bring forward to the next release level the custom modifications that you made to system data structures, run the JD Edwards EnterpriseOne Object Librarian Modifications report (R9840D) to list all of the modified data structures. Use this report as a guide when you manually enter data structure changes.

Business Functions

For any new custom business functions (BSFNs), create a new custom parent DLL to store your custom modifications. Always use the standard application program interface (API), jdeCallObject, to call other business functions from within a business function.

To bring your custom changes forward to the next release level, run the JD Edwards EnterpriseOne Object Librarian Modifications report (R9840D) to list all of the modified business functions. Use this report as a guide when you manually enter the business function changes.

To determine whether the source code of existing base business functions has been modified, use a third-party source compare tool, such as Microsoft WinDiff. To determine modifications to APIs within business functions, see the online help feature for the most current information about APIs.

This table describes the business function elements that are preserved or replaced during an upgrade:

Object	Preserved	Replaced	Comments
New custom business function objects.	X		
Modifications made to existing EnterpriseOne business function objects.		X	Named event rule (NER) BSFNs can be modified.

Versions

For new custom versions, create a new version with a name that does not begin with XJDE or ZJDE.

This table describes the versions elements that are preserved or replaced during an upgrade:

Object	Preserved	Replaced
Non-Oracle versions.	X	
Processing option data for ZJDE versions.	X	
Processing option data for XJDE versions.		X

In addition, processing option data is copied but not converted for non-Oracle versions that use processing option templates. A warning is issued at runtime, and some data might be lost.

Also, event rule modifications for custom versions of JD Edwards EnterpriseOne templates are not reconciled with the parent template.

Business Services

This table describes the business services elements that are preserved or replaced during an upgrade.

Object	Preserved	Replaced	Comments
New custom business service	X		Custom objects are always preserved.
New object within an existing JD Edwards EnterpriseOne business service		X	JD Edwards EnterpriseOne objects are always replaced.
Changed object within an existing JD Edwards EnterpriseOne business service		X	JD Edwards EnterpriseOne objects are always replaced.

Object	Preserved	Replaced	Comments
Business services selected as web services	X		Within P9603, you select which business services will be exposed as web services. This selection is preserved.

User Defined Objects

This table describes the User Defined Object elements that are preserved or replaced during an upgrade.

Object	Preserved	Replaced	Comments
New custom User Defined Objects	X		Custom objects are always preserved.
Modifications made to existing User Defined Objects		X	JD Edwards EnterpriseOne objects are always replaced.

4 Assembling Packages

Understanding the Package Assembly Process

The first step in building and deploying a package is to assemble the package. This includes entering a package name and detailed description, selecting the type of package that you want to build, and assembling the objects, foundation, and features that you want to include in the package. The package name and description appear during workstation installation when the user selects a package to install.

The Package Assembly Director, which you access from Oracle's JD Edwards EnterpriseOne Package Assembly program (P9601), steps you through the process. During package assembly, the build status is always either In Definition or Definition Complete. After you assemble the package, you can then define its build process.

Starting with Tools Release 9.2.5, a web version of package assembly has been created which you can access through Oracle's JD Edwards EnterpriseOne Package Assembly web application P9601W.

Package Assembly Director

The Package Assembly Director guides you through the process of specifying or confirming the location where package components can be found, as well as indicating the objects to include in the package. The director always gives you the option to either continue to the next form or go back to the previous form. Also, you can always cancel the assembly process.

You can enter default information on each of the main forms of the Package Assembly Director, or you can access subforms from each of the main forms to configure the information. The steps are the same whether you are adding components for the first time or revising a previously assembled package.

You can also access any previously assembled packages and view information about these packages by clicking the plus (+) symbol of the package on the Work with Packages form. For any previously assembled packages, underneath the package name you can view the package properties (including package type and current status), as well as the selections for foundation and language.

When you finish adding the selected components to a package, those components appear on the specific form for that component, the Package Component Revision form, and the Work with Packages form of the Package Assembly Director.

This table summarizes the function of each form in the Package Assembly Director:

Form	Function
Package Assembly Directory form	Use this form to review introductory information about the Package Assembly Director.
Package Information form	Use this form to enter the package name, description, and corresponding path code.
Package Type Selection form	Use this form to indicate whether you are creating a full or update package. When you create an update package, you must also indicate the parent package on which the update package is based. For example, if you were creating a package to update your original package called APPL_B, you would enter APPL_B as the parent package for your update package.

Form	Function
Foundation Component form	Use this form to enter the location of the foundation. A foundation is the code that is required to run all applications. It is required for all full packages. If you do not specify a foundation path for full packages, the system uses the default foundation path. Update packages use the foundation for the parent package unless you specify another foundation.
Database Component form (Prior to Tools Release 9.2.5.0)	Use this form to specify the location of the database to be included in the package. For full packages, if you do not specify a database location, the system uses the default database path. Update packages do not require a database.
Default Object Component form (for full packages only)	Use this form to verify the deployment data source. When you build a full package, the system retrieves the objects that are included in the package from the deployment data source that is associated with the path code that you specified for the package.
Object Component form (update packages only)	Use this form to specify the individual objects that you want to include in the package. You can add any of these objects: • Interactive or batch applications • Business functions • Business views • Data structures • Media object data structures • Table definitions
Features Component form	Use this form to include features in your package. A feature is a set of files or configuration options, such as registry settings, that must be copied to a workstation or server to support an application or another feature.
Language Component form	Use this form to include in your package language specifications for a language other than English.
Package Component Revisions form	Use this form to review the information that you entered on the previous forms. You can modify any or all of your selections on this form.

Accepting Default Values

Many of the forms in the Package Assembly Director have default values and, after verifying that you want to use the default value, you can advance to the next form without entering anything.

Forms determine the default values based on these criteria:

- Foundation

  The default foundation location is the server share path under the path code for the package.
- Database (Prior to Tools Release 9.2.5.0)

  The default database location is the server share path under the path code for the package.

- Objects

  The default location for full packages is the deployment data source.
- Language

  The default language is English.

On forms that have a default value, even if you change or clear the field you can always restore the original default value by clicking the Default button. The Form menu of the Package Component Revisions form also has a Set Default option that restores default values.

If you are building a full package and do not need to specify the objects in that package, the fastest way to define the package is to accept the default locations for the foundation and language. This method applies only to full packages. For an update package, if you accept the defaults but do not include any objects, the system creates an empty package.

As you view the forms in the Package Assembly Director, you can accept the default selections by clicking Next. If necessary, you can always make changes at the final Package Component Revisions form.

Verifying a Path Code for Package Assembly

Before you assemble a package, you can verify that the path code from which the package is built is configured correctly.

This section provides an overview of the process to verify a path code and discusses how to verify a path code for package assembly.

Understanding the Process to Verify a Path Code

The verification process tests the environment, machines, and tables before a package is submitted. By verifying your environment, you eliminate the chance that your package build will fail due to configuration issues. This verification can save many hours in building a package.

During the verification process, the program verifies these items:

- Disk space is adequate.
- Central objects and package build tables are accessible.
- User has permissions to create directories on the deployment server and enterprise server.
- Required service pack is installed.
- Machine tables are set up.
- Required compiler version is installed.
- Enterprise Server port is accessible.
- Debug levels of the jde.ini files are adequate for the client and enterprise server.

Form Used to Verify a Path Code for Package Assembly

Form Name	FormID	Navigation	Usage
Work with Batch Versions - Available Versions	W98305A	System Administration Tools, Package and Deployment Tools, Package Assembly Select Build Verification from the Form menu.	Verify the path code for package assembly.

Verifying a Path Code for Package Assembly

Access the Work with Batch Versions - Available Versions form.

1. Select a version and click Select.
2. On Version Prompting, click Submit.
3. Complete the Processing Options fields, and click OK.
4. On Printer Selection, select the desired printer, and click OK.

Assembling a Package Using Director Mode

Assembling a package in Director mode can involve configuring additional components, such as a foundation location or features. A feature is a set of files or configuration options, such as registry settings, that is copied to a workstation or server to support an application or other function. Like objects, features are built into a package and deployed to workstations and servers.

Note: Assembling a package in Express mode, rather than Director mode, is recommended in order to simplify and speed up the assembly process. Director mode can be used if you are unfamiliar with the process and would like to walk through each form consecutively. However, Express mode is the default mode for the Package Assembly application. This default behavior can be changed with a processing option.

This section discusses how to:

- Use Director mode to assemble a new package.
- Add a new foundation location.
- Add a database location. (Prior to Tools Release 9.2.5.0)
- Add features to a package.
- Review the package assembly selections.

Using Director Mode to Assemble a New Package

From System Administration Tools, select the Package and Deployment Tools menu, Package Assembly.

1. On the Work with Packages form, click Add, and then on the Package Assembly Director form, click Next.
2. On the Package Information form, complete the Package Name, Description, and Path Code fields.

   Note: The name of the package cannot be longer than eight characters.

   Note: You can build a single foundation package to deploy to all path codes by entering ***ALL** in the Path Code field. This option enables you to create an update package for service packs that can be installed to any path code. If you enter ***ALL** in the Path Code field, the application does not enable you to select, build, or deploy any objects (such as specs, business functions, and so on) in the package; you can only deploy a foundation. The package is built to a directory called all_packages, which is located under the release path (for example, e920/all_packages). This package can be deployed to any path code. Before you can use this option, you must define the *ALL path code in the Path Code Master application.

   See *"Setting Up Path Codes" in the JD Edwards EnterpriseOne Tools System Administration Guide*
3. Select the Director option and click Next.
4. On Package Type Selection, complete these fields, and click Next:

Field	Description
Full	Select to create a full package that contains all specifications and foundation code.
Update	Select to create an update package with specific items that can be deployed to specific users. If you are building a foundation package to the *ALL path code, the application automatically selects an update package.
Parent Package	Indicate the parent package that the update package is based on or related to. This information is used by the system to determine how to build business functions.
Build Business Service	Select to build the business server java files during the client package build. Client package must be selected to build business services.

5. On the Foundation Component form, perform one of these actions:
   - For full packages, accept the default location by clicking Next, or click Browse to specify another foundation location.
   - For an update, click Clear unless the package includes a foundation, and then click Next.

   See *Adding a New Foundation Location*.
6. On the Database Component form, perform one of these actions (Prior to Tools Release 9.2.5.0):
   - For full packages, accept the default location, or click Browse to specify another database location.
   - For an update, click Clear unless the package includes a foundation.

See *Adding a Database Location*.

7. Complete one of these actions:
   - If you are assembling a full package, click Next.

     For a full package, the system builds your package from the deployment data source that is associated with the default object path. Verify that the correct location appears on the form, and proceed to the next step.
   - If you are assembling an update package, click Next on the Database Component form, and then proceed to the next step.
8. On the Object Component form, to add an object, click Browse.

   The Object Component form only appears when you are assembling an update package. If you are assembling a full package, the Default Object Component form appears and you cannot add objects.
9. On the Object Component Selection form, locate and select the objects that you want to include in your update package, and then click Close to return to the Object Component form.

   When you revise a previously assembled package, objects that you added earlier also appear on the Object Component Selection form.
10. On the Object Component form or Default Object Component form, click Next.

    The Features Component form appears, on which you can specify the features that you want to include in your package. When you revise a previously assembled package, the system displays the features that you added earlier.
11. To add a feature, click Browse.
12. On the Feature Component Selection form, click Find to display a list of features, select one or more features, and then click Select to add the features that you want to include in your package.

    To select multiple features, press the Ctrl or Shift key while clicking features, and then click Select.
13. Click Close to return to the Features Component form.

    See *Adding Features to a Package*.
14. On the Feature Component form, click Next.
15. On the Language Component form, click Next if English is the only language that you want to configure.
16. To add a language to the language specifications for your package, double-click its row header in the detail area, and click Next.

    If you add a language to your package, only that language will be included. For example, if you add French, English will not be included even though it is the default language. To include two languages (such as French and English), you must select the detail records for both languages.
17. Continue with the task Reviewing the Package Assembly Selections.

Adding a New Foundation Location

Foundation items are automatically inserted into the database by the server manager when users installs a Tools Release. To add a new foundation, access the Foundation Item Revisions form by clicking Add on the Foundation Component Selection form.

1. Enter a foundation ID in the Foundation Name field.

   This is the code that is required to run all applications. A foundation ID is required for all full packages. For full packages, if you do not select a foundation, the default foundation is used. The default foundation is determined through the release that is associated with the path code for the package. This is normally the system directory at the same directory level as your path code. The foundation must be compressed when built.

2. Enter a service pack number in the Service Pack Number field, if appropriate.

   A service pack is an update to the foundation code that is delivered between major releases and cumulative releases of the software.
3. Enter the release number with which this foundation is associated in the Release field.
4. Enter the bitness of the system for this foundation.
5. Enter the host machine type in the Platform Type field.
6. Enter the compiler configuration to use for the software build in the Build Type field.
7. Enter the current status of the build process for foundation in the Foundation Build Status field.
8. Enter the date that the software build finished in the Date Built field.
9. Enter the time at which the software build finished in the Time of Build field.
10. Enter the name of the deployment server where your custom foundation resides in the Foundation Machine Key field.
11. Enter the exact path from which this item should be copied in the Foundation Path field.

    All files in the last directory that is specified will be included in the package. Source Machine Key and Source are used together to define the item's location.

Adding a Database Location

Access the Database Component Selection form.

1. Click Add to add a new database component.
2. Enter the name of the database component in the Database Name field.
3. Enter the name of the machine on the network (server or workstation) in the Source Machine Key field.
4. Enter the shared directory for this path code in the Server Share Path field.

Note: For full packages, if you do not specify a database location, the system uses the default database path (\pathcode\Packages\Data). Update packages do not require a database.

Adding Features to a Package

Access the Feature Component Selection form. On the Default Object Component form, click Next.

1. Find and select the existing features that you want to include in the package, and then click Select.

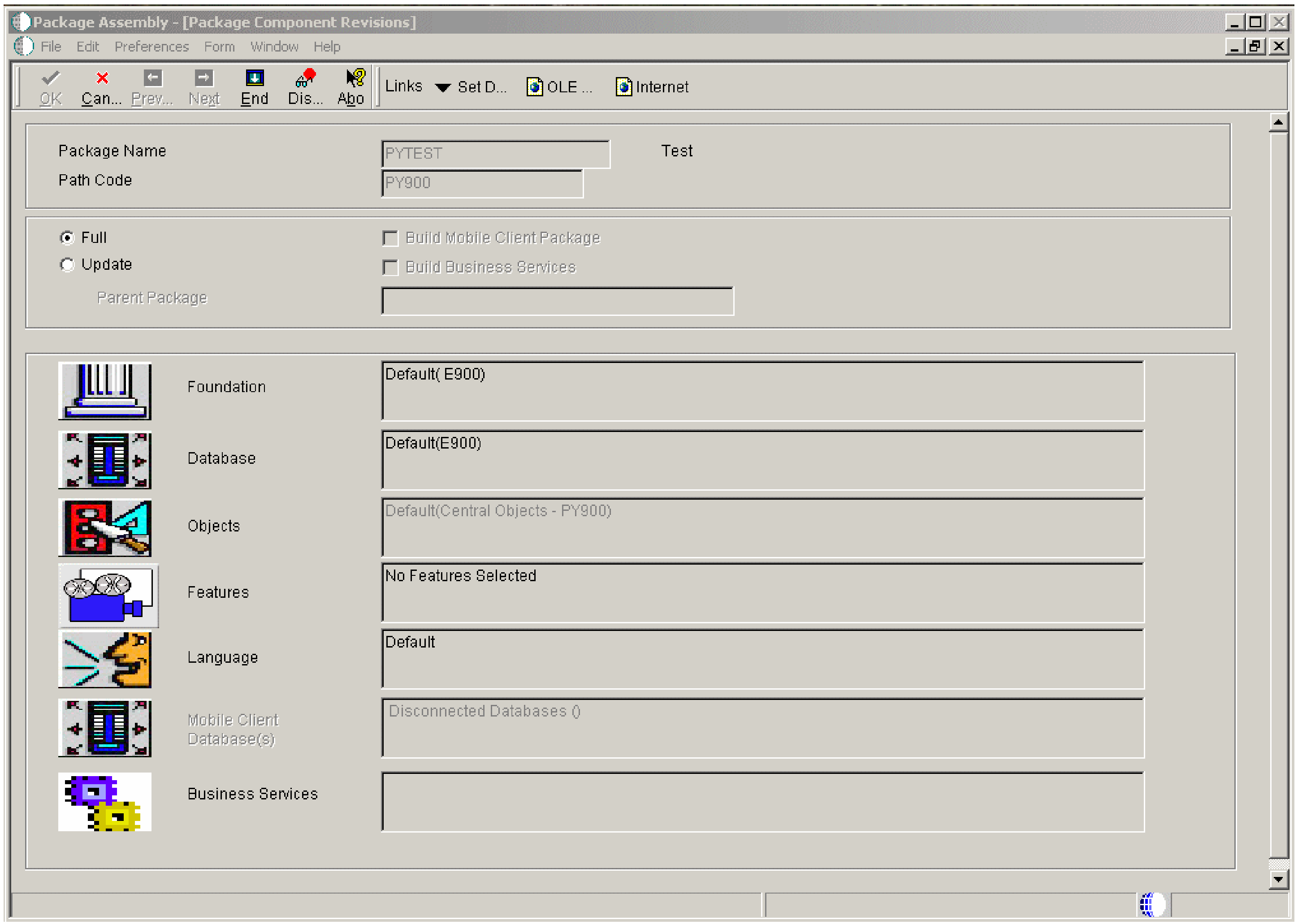
   To select multiple features, press the Ctrl or Shift key.
2. If the feature that you want to include has not been defined, you can create the feature definition by clicking Add.

   Oracle's JD Edwards EnterpriseOne Feature Based Deployment Director launches. You can use this director to create the new feature.

   See *Incorporating Features into Packages*.
3. Repeat steps 1 and 2 until you have finished adding features to your package.
4. When you are finished, click Close to return to the form from which you accessed the Feature Component Selection form.

Reviewing the Package Assembly Selections

Access the Package Component Revisions form. On the Language Component form, click Next.


Package Assembly - [Package Component Revisions]
File Edit Preferences Form Window Help
OK
Can...
Prev...
Next
End
Dis...
Abo
Links
Set D...
OLE ...
Internet

Package Name PYTEST Test

Path Code PY900

(● Full) Update

Build Mobile Client Package

Build Business Services

Parent Package

Foundation	Default(E900)
Database	Default(E900)
Objects	Default(Central Objects - PY900)
Features	No Features Selected
Language	Default
Mobile Client Database(s)	Disconnected Databases ()
Business Services	

1. Review the current foundation, database locations, objects, features, languages, and business services that exist for your package.

2. To change any of the package components, click the button for the component that you want to change.

   The form for that package component appears.
3. When you are finished assembling the package, click End to quit the Package Assembly Director.
4. Continue with the task Activating an Assembled Package.

Assembling a Package Using Express Mode

This section provides an overview of Express mode and lists the forms used to assemble a package using Express mode.

Understanding Express Mode

Express mode enables you to accept default values for the package assembly and then selectively choose which forms to view and modify. This may be preferable if you are familiar with the JD Edwards EnterpriseOne Package Assembly application and do not want to view and click Next through all of the Package Assembly forms.

When you select Express mode, you access the Package Component Revisions form, from which you can access the appropriate forms for the components that you want to update.

The JD Edwards EnterpriseOne Package Assembly application (P9601) is in Express mode by default. This can be changed to Director mode through a processing option.

See *Using Director Mode to Assemble a New Package*.

Forms Used to Assemble a Package Using Express Mode

Form Name	FormID	Navigation	Usage
Package Information	W9601F	Package and Deployment Tools (GH9083), Package Assembly Click Add, and then click Next.	Assemble a new package.
Package Component Revisions	W9601B	Select Express, and click Next on the Package Information form.	Review and revise the components in your package.
Foundation Component Selection	W9601H	Click the Foundation button on the Package Component Revisions form.	Add an existing foundation location to your package. Locate the existing foundation location that you want to use in the package, and click Select.

Form Name	FormID	Navigation	Usage
Foundation Item Revisions	W9883D	Click Add on the Foundation Component Selection form.	Add a new foundation location to the package.
Database Component Selection (Prior to Tools Release 9.2.5.0)	W9601N	Click the Database button on the Package Component Revisions form.	Add an existing database location to the package. Locate the existing database location that you want to use in the package and click Select.
Database Item Revisions (Prior to Tools Release 9.2.5.0)	W9883K	Click Add on the Database Component Selection form.	Add a new database location to the package.
Object Component Selection	W9601D	Click the Objects button on the Package Component Revisions form. Click Browse.	Add an object to the package.
Feature Component Selection	W9601AB	Click the Features button on the Package Component Revisions form. Click Browse.	Add a feature to the package.

Revising an Existing Package

This section provides an overview of the package revision process, lists prerequisites, and discusses how to revise an existing package.

Understanding the Package Revision Process

After you have assembled a package, you can use the Package Component Revision form to revise any of the components in the package. You do not need to complete all of the forms in the Package Assembly Director to revise a package.

Prerequisite

Verify that the status of the package is **In Definition.** If you try to revise a package that has a status of **Assembly Definition Complete,** the system displays an error message. To change the status of a package, select Active/Inactive from the Row menu on the Work with Packages form. You cannot revise or delete a package that has already been deployed.

Form Used to Revise an Existing Package

Form Name	FormID	Navigation	Usage
Package Component Revisions	W9601B	Package and Deployment Tools (GH9083), Package Assembly Select the package that you want to revise, and then select Package Revisions from the Row menu.	Revise an existing package.

Revising an Existing Package

Access the Package Component Revisions form.

1. Make any necessary changes to the package components.
2. When you are finished revising the package definition, click OK to return to the Work with Packages form.

   If any build information exists for the package, the system warns you that the changes will delete the existing build information.
3. Click one of these buttons:
   - OK

     Accept the revisions and delete the existing build information. If you accept the revisions, you should update the build information so that it reflects the changes that you made.
   - Cancel

     Delete the revisions and save the existing build information.

Activating an Assembled Package

This section provides an overview of the activation process and lists the forms used to activate an assembled package.

Understanding the Activation Process

After you have assembled a package, the package status remains at Assembly. While you define the package, it is inactive. You must activate the package to change the package status to Assembly-Definition Complete. An assembled package cannot be built until the status has been changed to Assembly-Definition Complete. The Assembly-Definition Complete status indicates that you are finished assembling the package and are ready to begin the build definition process.

You can change the package status at any time until you start the build definition process. That is, even after you have changed a package status to Assembly-Definition Complete, you can change the status back to In Definition if you need to revise the assembled package. When you are ready to define the build for the package, follow the steps described in Defining Package Builds.

Form Used to Activate an Assembled Package

Form Name	FormID	Navigation	Usage
Work with Packages	W9601L	Package and Deployment Tools (GH9083), Package Assembly Select the package that you want to activate, and select Active/ Inactive from the Row menu.	Activate the package. You can use this same process to change the status of a complete package back to In Definition.

Assembling a Package Using Web Applications (Release 9.2.5)

This section provides an overview of assembling a package using web applications and the forms used for this task.

Understanding Web Applications Used to Assemble Packages

Starting with Tools Release 9.2.5.0, you can assemble web packages using the JD Edwards EnterpriseOne Package Assembly web application (P9601W). This application enables you to call the server build and client build processes from the server.

Form Used to Assemble Packages Using Web Applications

Form Name	FormID	Navigation	Usage
Work with Packages	W9601L	Package and Deployment Tools (GH9083), Package Assembly	Activate the package. You can use this same process to change the

Form Name	FormID	Navigation	Usage
		Select the package that you want to activate, and select Active/ Inactive from the Row menu.	status of a complete package back to In Definition.

Assembling Packages Using Web Applications

1. Access the JD Edwards EnterpriseOne Package Assembly web application (P9601W).
2. Click Search to see the available packages.
3. Select the package you want to add and click Add.
4. Select OK.

   The details of the new package will be displayed on the right.
5. To change the settings, from the Row menu, select Language, Features or Foundation.

   Note: You can also add objects from the Row menu to update the package.
6. To activate the package, select Define Build from the Row menu.

5 Understanding the Package Build Process

How the System Builds Packages

This section discusses:

- Understanding Package Build with 64-bit (Release 9.2.3)
- Building 64-bit Packages for Business Function Development (Release 9.2.3)
- How the JD Edwards EnterpriseOne system builds a full server and client package.
- How the JD Edwards EnterpriseOne system builds an update server and client package.
- How the JD Edwards EnterpriseOne system builds a full client-only package.
- How the JD Edwards EnterpriseOne system builds an update client-only package.
- How the JD Edwards EnterpriseOne system deploys the full server package.
- How the JD Edwards EnterpriseOne system deploys the server update package.

Understanding Package Build with 64-bit (Release 9.2.3)

The EnterpriseOne system can be either a 32-bit system or 64-bit system. The package can build with either of these systems, but also depends on what the Enterprise Server system bitness is. The Enterprise Server can only build in the bitness that the system is running. If the Enterprise Server is running a 32-bit system, then the Server Only package will build a 32-bit package. If the Enterprise Server is running a 64-bit system, then the Server Only package will build a 64-bit package.

For a Server/Client package when the Enterprise Server is 32-bit, the client package must either point to a 32-bit system (multi-foundation) or the default system of the deployment server must be 32-bit. For a Server/Client package when the Enterprise Server is 64-bit, the client package must either point to a 64-bit system (multi-foundation) or the default system of the deployment server must be 64-bit.

(Prior to Tools Release 9.2.5) For a Client only package, the package can point to a 32-bit system or a 64-bit system using multi-foundation. In this case, it does not matter what bitness the Enterprise Server is.

The default system of the Deployment Server can be either a 32-bit or 64-bit system. It does not have to match the Enterprise Server, but the user will have to set up multi-foundation when selecting the foundation during package assembly. If the Enterprise Server is 64-bit and you are building a server/client package, then the foundation selected must be a 64-bit system. You select this in the Package Assembly application.

When building a package for the first time with a 64-bit foundation, the system starts a process to convert the include and source files in the repository, F98780R/F98780H, to 64-bit, include64, and source64. It inserts the include, source, include64, and source64 files into the repository. This is only done once. The F00942T-> emenvfu2 field is now set to 1 to indicate the files have been converted. Once this happens, when a package is built, the package contains both include, source, include64, and source64 files in the package. If the client foundation is 32-bit, then it uses the include and source to build bin32 and lib32 files with the include64 and source64 files existing in the package. If the client foundation is 64-bit, then it uses the include64 and source64 files to build bin64 and lib64 files with the include and source files existing in the package. If the Enterprise Server system is 32-bit, then it uses the checkin\include and checkin\source to build bin32 and lib32 files with the checkin\include64 and checkin\source existing in the package. If the Enterprise Server system is 64-bit, then it uses the checkin\include64 and checkin\source64 to build bin64 and lib64 files with the checkin\include and checkin\source existing in the package. During the client install, the include, source, include64, and source64 are delivered with the client package. During the deploy of the Enterprise Server, only the bin32 or bin64 are deployed. If the files were never converted, meaning a package was never built with a 64-bit foundation, then the package will not contain the 64-bit artifacts.

(Release 9.2.7.3) During the retrieval of files from the F98780R table, if a record is missing for an object or the files are missing from the par file, package build attempts to repair this record. Business functions and tables should always have a record in F98780R with a .h and/or a .c file. Business Service will also always have a record with a .java file. UBE and business views only have a record if they have a .h file. Application will only have a record if it has a bitmap file.

Understanding 64-bit Enablement with the Release Master (Release 9.2.4)

With Tools Release 9.2.4.0 and beyond presents the ability to have a 64-bit only system. The Release Master application is used to indicate a 64-bit only system. When you have all pathcodes, the Enterprise system, and the client foundation converted to 64-bit, and enabled 64-bit in the Release Master, the package will only contain include64 and source64. The package will build the dlls with a 64-bit settings. The client install will have bin64, obj64, lib64, source64 and include64. The server will deploy the bin64 directory. In Busbuild, you will be able to compile, link and build in 64-bit only. Once the 64-bit release is enable, there is no going back 32-bit.

If you update all the pathcodes to 64-bit and have enabled the Release flag for 64-bit the package will only extract and build the 64-bit artifacts, resulting in the following:

- When assembling the package, you can only select 64-bit foundations.
- During the retrieval of the .c and .h file on the Enterprise Server, package build will only extract the 64-bit files, include64 and source64.
- The building of business functions on the Enterprise Server will build the 64-bit artifacts for the 64-bit dlls or .so or pgm files.
- The process to copy the include and source to the Development Client will only transfer the checkin\ include64 and checkin\source64 to the Development Client.
- The client Busbuild will only build and compile the 64-bit artifacts to create the bin64, lib64 and obj64 files.
- Only the 64-bit directories will be compressed, include64, source64, bin64, obj64 and lib64.

Building 64-bit Packages for Business Function Development

If you are building a package to use in developing 64-bit business functions, you will need to select the Generate NER Headers for Opposite bitness check box in the Package Definition application. When you compile a 64-bit business function in OMW with Busbuild on a 32-bit development client the NER headers (n*.h) for 64-bit will have to be present on the client machine for the compile to be successful. This is done during the client package build prior to building the business functions.

When you define a package in the Package Definition application for building business functions, you must select the 'Generate NER headers for Opposite bitness' check box. This creates the NER headers (n*.h) for the opposite bitness under include64 or include.

If the package is a 32-bit package and the box is checked, then the headers for NER are generated into the include64 directory of the package.

If the package is a 64-bit package and the box is checked, then the headers for NER are generated into the include directory of the package.

The Generation of these NER headers are displayed under in the buildlog.txt, <deployment server>\<pathcode>\<package name>\work\buildlog.txt.

If you do not select the option to Generate NER headers for Opposite bitness during the package build, the NER headers for the opposite bitness are not present in the include or include64 directory. You will see these errors when compiling the opposite bitness in OMW with Busbuild.

Building Full Package in Net Change Mode (Release 9.2.9)

Starting with Tools Release 9.2.9, you can build net change objects for full packages. When you build net change objects for full packages, the system builds all specs and builds only the business functions, tables, UBE with files, and business view with files that changed through OMW or ESU process since the last active package. The system first copies the active package files, include, source, obj, bin32\bin64 and lib32\lib32 to the new package. Then, it builds only the business functions, named event rules (NER), and table event rules (TER) that changed. The system also builds all spec tables and when complete, compresses the files and sets up the deploy. Also, the system starts the client build at the beginning of the server build. This process substantially reduces the time it takes to build a full package which can then be deployed without locks.

Note: `Click here to view a recording of this feature.`

Note: `Click here to view an OBE of this feature.`

How the System Builds a Full Server and Client Package

This is an overview of how the JD Edwards Enterprise One system builds a full server and client package. The beginning of each step states whether the process occurs on the client machine or on the Enterprise Server.

Note: Any references to 64-bit apply to Tools Release 9.2.3 and higher.

1. Client: Server Package UBE (R9621S) initiates the server package build.
2. Client: The system creates the package build directories on the deployment machine.
3. Client: The system checks the F00942T->emdbsrcflg value for 1. If value is 0 or blank, the objects have not been inserted into the F98780R table. If value is 1, then the objects have been inserted.
4. Client: The system checks the F00942T-emenvfu2 value for 1. If the value is 0 or blank, then the include and source files in the F98780R/R98780H have not been converted to 64-bit. If the value is 1, then the include and source files have been converted to include64 and source64 and uploaded into the F98780R/R98780H tables
5. Client: This is only done once. If F00942T -> emdbsrcflg value is 0 or blank and the F00942T-> emenvfu2 value is 0 or blank, then the system sets F00942T -> emdbsrcflg=2 for processing. It does a fetch from F9860 for all objects that are not system code 88. It checks if there is a file associated with the object on the deployment server pathcode `include` and `source` directories. If there are files in the `include` and `source` directories, then it creates a par file for the object and insert the record with the par file into the F98780R/F98780H. If the foundation selected for the package was 32-bit, then for business functions and tables, the par file will only include source and include files. It sets the object record in the F9860->sigtffu =1 to indicate there are files for

this object. When the process is complete, it sets the F00942T->embdsrcflg=1 for completion. It leaves the F00942T-> emenvfu2 as 0 or blank since this indicates that the conversion of the include and source into 64-bit files was not done.

6. Client: This is only done once. If F00942T -> emdbsrcflg value is 0 or blank and the F00942T-> emenvfu2 value is 0 or blank, then the system sets F00942T -> emdbsrcflg=2 for processing. It does a fetch from F9860 for all objects that are not system code 88. It checks if there is a file associated with the object on the deployment server pathcode `include` and `source` directories. If there are files in the `include` and `source` directories, then it creates a par file for the object and inserts the record with the par file into the F98780R/F98780H. If the foundation selected for the package was 64-bit, then for business functions and tables, it converts the include and source into 64-bit files into a temporary include64 and source64 directory. It creates the par file with all four directories and inserts the record into the F98780R/F98780H . It sets the object record in the F9860->sigtffu =1 to indicate there are files for this object. When the process is complete, it sets the F00942T->embdsrcflg=1 for completion. It also sets the F00942T-> emenvfu2 =1, which indicates the conversion of the include and source into 64-bit files was done.
7. Client: This is only done once. If F00942T -> emdbsrcflg value is 1, the F00942T-> emenvfu2 value is 0 or blank, and the foundation selected for the package was 64-bit, then it sets F00942T -> emenvfu2 =2 for processing. It does a fetch from F9860 for all objects that have F9860-> sigtffu=1 and that are not APPL, GT or DSTR, and not system code 88. For each fetch, it retrieves the par file from the F98780R, expands the par file, converts the include and source files to 64-bit and places them in temporary include64 and source64 directories. It creates the par file with all four directories and inserts it into the repository. When complete, it sets the F00942T-> emenvfu2 =1 to indicate the files have been converted to 64-bit.
8. Client: The system initiates the connection with the Primary Enterprise Server.
9. Enterprise Server: Creates the directories on the Primary Enterprise Server and builds the list of objects from the F96225 table.
10. Enterprise Server: If client populated the F98780R/F98780H, then the system checks the 'Live' specs F98710 <package name> for the object names F98780R, F98780H, and F00942T. If these specs are not there, then it get the specs from the Central Objects path code and insert them into the 'Live' specs tables, F98710, F98711, F98712 and F98713.
11. Enterprise Server: If F00942T -> emdbsrcflg= 1, open the F9860 and get all objects where a F9860->sigtffu1=1 and F9860->sy != 88 and not a Named Event Rule (NER) or a Business Service (SBF) object. Open the F98780R and retrieve each object which will put the files into Primary Enterprise Server directories, checkin\source (.c files), checkin\source64 (.c files), checkin\include(.h files) , checkin\include64(.h files) and the bitmaps into the checkin\res directory.
12. (Tools Release 9.2.7.3) Enterprise Server: The system validates each BSFN and TBLE if there is a record in the F98780R table, the record has a par file, and the .h and/or .c file exist in the checkin location.
13. (Tools Release 9.2.7.3) Enterprise Server: The system validates each UBE, BSVW, and APPL with F9860->sigtffu1=1 if there is a record in the F98780R table, the record has a par file and for a UBE and BSVW, if the .h file exists in the checkin location and for an APPL, if a bitmap exists.
14. (Tools Release 9.2.7.3) Enterprise Server: If the validation fails and the Repair Repository flag is set, then the system inserts a record into the F96225 table with package name, pathcode, primary server name, object name, version = “REPAIR”, operation = “11”, and date and time.
15. (Tools Release 9.2.7.3) Enterprise Server: The system checks if any BSFN or TBLE does not have F9860->sigtffu1=1 and is checked in. If found, the system inserts a record into the F96225 table with package name, pathcode, primary server name, object name, version = “REPAIR” and option = “11”, and date and time.
16. Client: It compresses the res directory from the check-in location of the deployment server. Transfers the res.cab to the Primary Enterprise Server under the checkin\res folder.
17. (Tools Release 9.2.7.3) Client: If the Repair Repository flag is set then the system selects F96225 records where package name, pathcode, primary server, operation=”11”, and status = ERROR.

    For each record, the system attempts to get the record from the F98780R table, evaluate what is missing. If the record is not in the F98780R table, the system recreates the par file and inserts the record into the F98780R and F98780H tables.

18. (Tools Release 9.2.7.3) Enterprise Server: If the Repair Repository flag is set then the system selects the F96225 records where package name, pathcode, primary server, operation=”11”, and status = ERROR. The system then retreives and validates each record in the F98780R table if there is a .h or/and .c file. If successful, then the system sets the F96225 record to status=SUCCESS.
19. Enterprise Server: If there are other servers, the Primary Enterprise Server will contact each Enterprise Server to start the build process.
20. Enterprise Server: On the Primary Enterprise Server, the package build checks if there is a difference between data dictionary items in the database compared to what is in ddict/ddtext. The difference fields are the type and length. If there are any differences, then it locks all processes to prevent any UBE to run and waits for all UBEs to finish. It does not affect users on the Web. When the locks are obtained, ddict/ddtext/glbtb are deleted in the path code spec directory.
21. Enterprise Server: The Named event rules (NERs) are generated into the checkin\source and checkin\include directories if the system on the Enterprise Server is 32-bit. If the system on the Enterprise Server is 64-bit then it will generated the files into the checkin\source64 and checkin\include64 on the Primary Enterprise Server
22. Enterprise Server: Moves server-only .c and .h files (if 32-bit system moves the checkin include and source, if 64-bit system moves the include64 and source64) to the source and include directories respectively on the Primary Enterprise Server. If there is more than one server, then the Primary Enterprise Server moves the .c and .h files from the source and include directories to the other Enterprise Servers under the source and include directories.
23. Enterprise Server: Compiles business functions on the Primary Enterprise Server to generate .dll, .so, .sl, or .SRVPGM files. If there are multiple servers, the Primary Enterprise Server sends a message to the other servers to start the build of business functions.
24. Enterprise Server: IBM i (Starting with Tools Release 9.2.6.0) A new unique library is created in which you can create the SVRPGM, MODULES and FILES. This unique library, is created with the first 2 letters and the last 2 letters of the package, and the job number. For example, if the package is DV920FA, then the unique library is created as "DVFA000340". This name is stored in the F9603 file in phpkgdeffut3.
25. Enterprise Server: Builds specs on the Primary Enterprise Server from Central Objects, placing the results in the package's spec tables, <table name>_<package name> (for example, F98710_DV920FA) in the database. There are 17 tables that are copied from one location to the other. Also the manifest table is created which holds the name of the package.
26. Enterprise Server: If the client package was also selected, the Primary Enterprise Server transfers the files from the checkin\source, checkin\source64, checkin\include, checkin\include64 and checkin\res to the deployment server package include, source, include64, source64 and res directories.
27. Enterprise Server: If client package was selected, it transfers the res.cab from the checkin\res to the package directory. The client then uncompresses it into the <package name>\res directory.
28. Client: Send message to Primary Server to get status.
29. Enterprise Server: Checks the status of the other servers to see if BSFNs are complete. Checks its own status to see if BSFNs are complete.
30. Enterprise Server: When the dll and specs are finished, if the compress feature for server is enabled, it compresses the .dll, .so, .sl, and .SRVPGM files on the Primary Enterprise Servers. The Primary Enterprise Server sends a message to all other servers to compress
31. Enterprise Server: Transfers the compressed files and the compress.inf file back to the deployment server under the <packagename>/<machine type> directory.
32. (Starting with Tools Release 9.2.6) UNIX/Windows: Under the deploy location, creates the package name under the bin64 and specs folders in these paths, respectively: <pathcode>\bin64\<package name> and <pathcode>\spec\<package name>.
33. Starting with Tools Release 9.2.6) IBMi: Creates a new library with the package name.
34. Starting with Tools Release 9.2.6) IBMi: Copies from the unique library the SVRPGM, MODULES and FILES to the package name library.
35. (Starting with Tools Release 9.2.6) UNIX/Windows: Copies the package bin64 directory to the <pathcode>\bin64\<package name> directory.

36. (Starting with Tools Release 9.2.6) Under <pathcode>\spec\<package name>, creates the spec.ini file with the database datasource location for Central Objects.
37. Enterprise Server: Check the jde.ini for entry

    ```
    [PACKAGE MANIFEST]   CreateJAR=Y
    ```

    (Tools Release 9.2.7) and/or check the F96215->PKGBULFUT3 = "Y" which is set when you select **Compress Application Component** in the compress options tab.

    If this entry is in jde.ini or F96215 with 'Y', then create a E1_APP_xxxx.jar file under the package name directory on the Enterprise Server. This jar file contains the bin64 directory and the spec.ini populated with the package name. The jar file is used by the Server Manager to create a new Enterprise Server Instance.
38. Enterprise Server: Transfers the server log (svrpkgbuild.log) from each server to the deployment server under <packagename>\serverLogs directory.
39. Enterprise Server: Transfers the files located under the text directory on the each Enterprise Server to the Deployment Server under <packagename>\serverlogs\<server name>\text.
40. Enterprise Server: Transfers the files located under the CompileLogs directory on the each Enterprise Server to the deployment server under <packagename>\serverlogs\<server name>\CompileLog.
41. Enterprise Server: Transfers the files located under the GenerateNER_logs directory on the Primary Enterprise Server to the deployment server under <packagename>\serverlogs\GenerateNER_logs.
42. If this is a server only package, the process is complete. However, if client was selected, the system performs the next steps.
43. Client: Runs Client Package UBE R9622C to initiate the client package build.
44. Client: Creates the package inf file.
45. Client: Compiles the .c and .h files using Busbuild.
46. Client (Tools Release 9.2.5): Creates the spec.ini file that contains the package name and the Central Objects data source. The spec.ini file points to specs of the package tables in the Central Objects data source. The local spec database is no longer created.

    Client (Prior to Tools Release 9.2.5): Builds the specs from the package's spec tables <table name>_<package name> (for example, F98710_DV920FA) in the database and puts them into the package's local spec database.
47. Client: If F00942T-> emdbsrcflg= 1 or 3 then get all the SBF files from the F98780R repository table. If the field was blank or 0 then copy the files from <pathcode>\sbf directory. Build the BSSV - Business Services.
48. Client: Waits for the compile of the business functions, building of the specs, and the building of the BSSV to complete.
49. Client: Copies the generated NER .c and .h files back to the check in location.
50. Client: Compresses the directories on the Deployment Server if compression was selected.
51. Client and Server package is complete. The User can deploy the Server package.

How the System Builds an Update Server/Client Package

This is an overview of how the JD Edwards EnterpriseOne system builds an update server and client package. The beginning of each step states whether the process occurs on the client from where the build is performed or on the Enterprise Server.

Note: Any references to 64-bit apply to Tools Release 9.2.3 and higher.

1. Client: Server Package (R9621S) initiates the server package build.
2. Client: The system creates the package build directories.
3. Client: The system initiates the connection with the Primary Enterprise Server.
4. Enterprise Server: Creates the directories on the Primary Enterprise Server.

5. Enterprise Server: The system builds a link list of objects in the update package from the F96225 table. If F00942T-> emdbsrcflg = 0 or blank, then it builds the list of business functions, NER, and Table ER in a file called "bsfnfile.txt". A list of bitmaps are listed in a file called "bitmapfile.txt". Both files are transferred from the <package name>\work directory to the Deployment Server.
6. Client: If F00942T-> emdbsrcflg = blank or 0 then the system opens the "bsfnfile.txt" for the list of .c and .h files to transfer from the deployment server check-in location to the Primary Enterprise Server directories, checkin \source (.c files) and checkin\include (.h files). This is done to preserve a snapshot of the .c and .h files at the time of the build.
7. Client: If F00942T-> emdbsrcflg=blank or 0, then system opens the "bitmapfile.txt".for the list of bitmaps to transfer from the res directory check in location on the deployment server to the Primary Enterprise Server checkin\res directory.
8. Enterprise Server: If F00942T->emdbsrcflg=1, then the system opens the link list of Objects and if the F9860->sigtffu1, it retrieves the par file from the F98780R and puts them in the checkin\include, checkin\include64, checkin\source, checkin\source64 and checkin\res directories.
9. (Release 9.2.7.3) Enterprise Server: For each object, the system validates each BSFN and TBLE if there is a record in the F98780R table, the record has a par file, and the .h and/or .c file exists in the checkin location.
10. (Release 9.2.7.3) Enterprise Server: The system validates each UBE, BSVW, and APPL with F9860->sigtffu1=1 if there is a record in the F98780R table, the record has a par file and for a UBE and BSVW, a .h file exists in the checkin location and for an APPL, a bitmap exists.
11. (Release 9.2.7.3) Enterprise Server: If the validation failed and the Repair Repository flag was set, then the system inserts a record into the F96225 table with package name, pathcode, primary server name, object name, version = “REPAIR”, operation = “11”, and date and time.
12. (Release 9.2.7.3) Enterprise Server: For each UBE, BSVW, and APPL which does not have F9860->sigffu1=1, the system inserts a record into the F96225 table as a REPAIR record. The client checks if there is a file associated with it and repairs it.
13. (Release 9.2.7.3) Client: If the Repair Repository flag is set, the system selects F96225 where package name, pathcode, primary server, operation=”11”, and status = ERROR.

    For each record retrieved from the F98780R table, the system evaluates what is wrong or missing and recreates the par file. The system then inserts the record back in the F98780R and F98780H tables.
14. (Release 9.2.7.3) Enterprise Server: If the Repair Repository flag is set, then the system selects F96225 records where package name, pathcode, primary server, operation=”11”, and status = ERROR. The system then retrieves and validates each record from the F98780R table if there is a .h or/and .c file or a bitmap. If the validation is successful, the system sets the F96225 record to status=SUCCESS.
15. Enterprise Server: On the Primary Enterprise Server, if the objects in the update package contain NER, on the Primary Enterprise Server, the package build checks if there is a difference between data dict items in the database compared to what is in ddict/ddtext. The difference fields are the type and length. If there are any differences, then it locks the package build locks all processes to prevent any UBE to run and waits for all UBEs to finish. It does not affect users on the Web. When the locks are obtained, ddict/ddtext/glbtb are deleted in the path code spec directory. The locks are then released.
16. Enterprise Server: The system generates named event rules (NERs) on the Primary Enterprise Server. The generated files are created either in the checkin\source and checkin\include directories or checkin\source64 and checkin\include64 directories depending on the bitness.
17. Enterprise Server: The Primary Enterprise Server copies server only .c and .h files to the source and include directories respectively. If there are more than one server, then the Primary Enterprise Server transfers the .c and .h files to the other Enterprise Servers under the source and include directories.

    (Starting with Tools Release 9.2.4.0) If 64-bit is enabled in Release Master, only the include64 and source64 artifacts are copied into the checkin location (Release 9.2.4).
18. Enterprise Server: Copies the parent dlls from the parent package to the update package. This dll is used in the compile of the business functions for the update package. If there are multiple servers, the Primary Enterprise Server sends a message to the other servers to start the build of business functions.

19. (Starting with Tools Release 9.2.6.0) IBMi: Opens the F9603 to get the unique library of the parent package from phpkgdeffut3. It copies SVRPGM to the update package library, and compiles the business functions and links with the library location of the parent package.
20. Enterprise Server: The Primary Enterprise Server builds specs from Central Objects, placing the results in the package's spec tables, <table name>_<package name> (for example, F98710_DV920FA) in the database. The 17 tables are created and the records for the objects are inserted into the new tables. The Manifest table is created with the name of the package and all the objects in the update package.
21. Enterprise Server: The Primary Enterprise Server creates the tables F989999 and F989998 in the Central objects location with the name F989999_<package name> and F989998_ <package name>.
22. Enterprise Server: If the client package was also selected, the Primary Enterprise Server transfers the files from the checkin\source, checkin\include, checkin\source64 and checkin\include64 to the deployment server package include, source, include64 and source64 directories.
23. Enterprise Server: If client package was selected, the Primary Enterprise Server transfers back res bitmaps from the checkin\res to the Deployment Server in the <package name>\res directory.
24. Enterprise Server: The system waits for specs to finish and for dlls to finish building. If there were other servers, it will wait for them to complete.
25. Enterprise Server: When the dlls and specs are finished building, if the compression was selected for the server, then the system compresses the .dll, .so, .sl, and .SRVPGM files on the Enterprise Servers.
26. Enterprise Server: The Primary Enterprise Server transfers the compressed files and the compress.inf file back to the Deployment Server under the <packagename>/<machine type> directory.
27. Enterprise Server: The Primary Enterprise Server transfers the server log (svrpkgbuild.log) from each server to the Deployment Server under <packagename>\serverLogs directory.
28. Enterprise Server: Transfers the files located under the text directory on each Enterprise Server to the Deployment Server under <packagename>\serverlogs\<server name>\text.

    (Starting with Tools Release 9.2.4.0) If 64-bit is enabled in Release Master, only the include64 and source64 artifacts are copied to the client and the res files to the Deployment Server.
29. Enterprise Server: Transfers the files located under the CompileLogs directory on each Enterprise Server to the Deployment Server under <packagename>\serverlogs\<server name>\CompileLogs.
30. Enterprise Server: Transfers the files located under the GenerateNER_logs directory on the Primary Enterprise Server to the Deployment Server under <packagename>\serverlogs\GenerateNER_logs.
31. If this is a server only package, the process is complete. However, if you have selected to build a client, the system also performs the next steps.
32. Client: The Client Package UBE R9622C initiates the client package build.
33. Client: The system creates the package inf file.
34. Client: The system compiles the .c and .h files using Busbuild. Busbuild copies the dll from the parent package to build the update business functions into the parent package dll.
35. Client (Starting with Tools Release 9.2.5.0): No specs are built. For each object a .par file is created under the spec\"package name" directory.

    Client (Prior to Tools Release 9.2.5.0): The system builds the specs from the package's spec tables <table name>_<package name> (for example, F98710_DV920FA) into TAM files located under the spec directory.
36. Client: The system waits for the business functions to compile and the spec build to complete.
37. Client: The system copies the generated NER .c and .h files back to the check-in location.
38. Client (optional): The system compresses the directories on the Deployment Server. This is only necessary if building an ESU. The client install process does not use the compressed update package files.

How the System Builds a Full Client Package (Prior to Tools Release 9.2.5)

The following sections provide an overview of how the JD Edwards EnterpriseOne system builds a full client-only package. All activity is done on the client/build machine.

Note: Any references to 64-bit apply to Tools Release 9.2.3 and higher.

1. Server Package UBE (R9621S) runs first and initiates the server package build. The process determines this is a Client Only package and will continue to the R9622C UBE.
2. Client Package UBE (R9622C) runs and initiates the client package build.
3. Creates the package build directories.
4. The system checks the F00942T->emdbsrcflg value for 1. If value is 0 or blank, the objects have not been inserted into the F98780R table. If value is 1, then the objects have been inserted.Client: This is only done once. If F00942T -> emdbsrcflg value is 0 or blank and the F00942T-> emenvfu2 value is 0 or blank, then the system sets F00942T -> emdbsrcflg=2 for processing. It does a fetch from F9860 for all objects that are not system code 88. It checks if there is a file associated with the object on the deployment server pathcode `include` and `source` directories. If there are files in the `include` and `source` directories, then it creates a par file for the object and inserts the record with the par file into the F98780R/F98780H. If the foundation selected for the package was 32-bit, then for business functions and tables, the par file only includes source and include files. It sets the object record in the F9860->sigtffu =1 to indicate that there are files for this object. When the process is complete, it sets the F00942T->embdsrcflg=1 for completion. It leaves the F00942T-> emenvfu2 as 0 or blank, as this indicates that the conversion of the include and source into 64-bit files was not done
5. This is only done once. If F00942T -> emdbsrcflg value is 0 or blank and the F00942T-> emenvfu2 value is 0 or blank, then the system sets F00942T -> emdbsrcflg=2 for processing. It does a fetch from F9860 for all objects that are not system code 88. It checks if there is a file associated with the object on the deployment server pathcode `include` and `source` directories. If there are files in the `include` and `source` directories, then it creates a par file for the object and inserts the record with the par file into the F98780R/F98780H. If the foundation selected for the package was 64-bit, then for business functions and tables, it converts the include and source into 64-bit files in temporary include64 and source64 directories. It creates the par file with all four directories and inserts the record into the F98780R/F98780H. It sets the object record in the F9860->sigtffu =1 to indicate there are files for this object. When the process is complete, it sets the F00942T->embdsrcflg=1 for completion. It sets the F00942T-> emenvfu2 =1, which indicates that the conversion of the include and source into 64-bit files was done.
6. This is only done once. If F00942T -> emdbsrcflg value is 1, the F00942T-> emenvfu2 value is 0 or blank, and the foundation selected for the package was 64-bit, then it sets F00942T -> emenvfu2 =2 for processing. It does a fetch from F9860 for all object that have F9860-> sigtffu=1 and that are not APPL, GT, or DSTR, and not system code 88. For each fetch, it retrieves the par file from the F98780R, expands the par file, converts the include and source files to 64-bit and places them in temporary include64 and source64 directories. It creates the par file with all four directories and inserts it in the repository. When complete, it sets the F00942T-> emenvfu2 =1 to indicate the files have been converted to 64-bit.
7. Creates the INF file.
8. If F00942T-> emdbsrcflg= 0 or blank, then it copies the files from the check-in location to the package name directory. These files are located under the directories res, source (.c files), include (.h files) and work.

   If 64-bit is enabled in Release Master, only the include64 and source64 artifacts are copied into the check-in location (Starting with Tools Release 9.2.4.0).
9. If F00942T->1, then it queries the F9860 for all objects with a F9860->sigtffu=1 and F9860->sy != 88. It retrieves each object from the F98780R and put them into the directories res, source (.c files), include (.h files),

If the package is pointing to a 64-bit foundation, then it will put files into the source64 (.c files) and include64 (.h files) directories. It then copies the work directory from the checkin location to the package name.

If 64-bit is enabled in Release Master, only the include64 and source64 artifacts are copied to the package location (Starting with Tools Release 9.2.4.0).

10. Creates an OEE Local database with the Central Object tables.
11. Copies the 17 Central Objects tables into the created database.
12. Runs the JD Edwards Enterprise One BusBuild program to generate the NER, compile and link the business functions, which create the DLLs in the bin32/bin64 directory, the objects in the obj directory, and the libraries in the lib32/lib64 directory.
13. Build Business Services if selected.
14. Build any Features that were included in the package.
15. Compress the directories.

How the System Builds Client Only Update Package (Prior to Tools Release 9.2.5)

The following section describes how the JD Edwards EnterpriseOne system builds a client only update package.

Note: Any references to 64-bit apply to Tools Release 9.2.3 and higher.

1. Server Package UBE (R9621S) runs first and initiates the server package build. The process determines this is a Client Only package and will continue to the R9622C UBE.
2. Client Package UBE (R9622C) runs and initiates the client package build.
3. Creates the package build directories.
4. Creates the INF file.
5. For each object in the Package Build History table (F96225), retrieves theinformation from the Central Objects and adds it to the TAM specification files and builds the individual .par file for the object under the <package name>\spec\packagename> directory.
6. If F00942T-> emdbsrcflg= 0 or blank, copies the associated .c, .h, and .hxx files for the selected objects from the check-in location on the deployment server to the package's directory on the Deployment Server.
7. If F00942T->emdbsrcflg=1, then it retrieves each par file from the F98780R and puts the files in the package's directory on the Deployment Server.
8. Runs the JD Edwards EnterpriseOne BusBuild program. Generates the NER from the list of objects. It copies from the parent package, the bin32/bin64 and lib32/lib64 files to the update package bin32/bin64 and lib32/lib64 directory. It compiles the business functions objects in the package. The updates are now in the bin32/bin64 and lib32/lib64 files of the update package.
9. Updates the Parent Package: If the build machine is not the deployment server, it will copy the parent package local database from the spec directory to the build machine. The attach and detach of the parent local database can only occur on a local machine and not across the network. The specs from the update package are than merged into this parent package local database. The parent package local database is then copied back to the deployment server parent package directory. If the build machine is the deployment server, than the merge of the update objects happens to the parent package specs located under the parent package spec directory.
10. Copies from the update package bin32/bin64, lib32/lib64, obj, source/source64 and include/include64 files to the parent package directory on the Deployment Server.
11. Builds any features included in the package.
12. If compression was selected, then compresses the directories.

How the System Builds a Full Server Package with Net Change Objects (Release 9.2.9)

The following section describes how the JD Edwards EnterpriseOne system builds a full server and client package with net change objects. The system builds all of the specs and the changed objects, business functions, tables, UBE with files, and BSSVW with files, since the last active package was built. The main files are copied from the active package to the new package and the changed objects are retrieved from the repository.

The beginning of each step states whether the process occurs on the Deployment Server or on the Enterprise Server. The client build uses the Deployment Server Services for the build.

1. Client: Server Package UBE (R9621S) initiates the server package build.
2. Client: Creates the package build directories on the deployment machine.
3. Client: If Build Net Change Objects is selected, sets the flag for net change mode.
4. Client: Checks the F00942T->`emdbsrcflg` value for `1`. If the value is `0` or blank, the objects have not been inserted into the F98780R table. If the value is `1`, then the objects have been inserted.
5. Client: Checks the F00942T->`emenvfu2` value for `1`. If the value is `0` or blank, then the include and source files in the F98780R/R98780H tables have not been converted to 64-bit. If the value is `1`, then the include and source files have been converted to include64 and source64 and uploaded into the F98780R/R98780H tables.
6. Client: This is a one-time process. If F00942T -> `emdbsrcflg` value is `0` or blank and the F00942T-> emenvfu2 value is 0 or blank, then the system sets F00942T -> `emdbsrcflg=2` for processing. It does a fetch from F9860 for all objects that are not system code 88.

   The client server checks if there is a file associated with the object on the Deployment Server pathcode `include` and `source` directories. If there are files in the `include` and `source` directories, then the system creates a par file for the object and inserts the record with the par file into the F98780R/F98780H tables.

   If the foundation selected for the package is 32-bit, then for business functions and tables, the par file only includes source and include files. The system sets the object record in F9860->`sigtffu =1` to indicate there are files for this object. When the process is complete, it sets F00942T->`embdsrcflg=1` for completion. The system leaves the F00942T-> `emenvfu2` as `0` or blank since this indicates that the conversion of the include and source files into 64-bit files is not done.
7. Client: This is a one-time process. If F00942T -> `emdbsrcflg` value is `0` or blank and the F00942T-> `emenvfu2` value is `0` or blank, then the system sets F00942T -> `emdbsrcflg=2` for processing. The system fetches from F9860 for all objects that are not system code 88. It checks if there is a file associated with the object on the Deployment Server pathcode include and source directories. If there are files in the include and source directories, then it creates a par file for the object and inserts the record with the par file into the F98780R/F98780H tables.

   If the foundation selected for the package is 64-bit, then for business functions and tables, it converts the include and source files into 64-bit files in temporary include64 and source64 directories. It creates the par file in all four directories and inserts the record into the F98780R/F98780H tables. It sets the object record in the F9860->`sigtffu =1` to indicate there are files for this object. When the process is complete, it sets the F00942T->`embdsrcflg=1` for completion. It also sets the F00942T-> `emenvfu2 =1`, which indicates that the conversion of the include and source files into 64-bit files is done.
8. Client: This is a one-time process. If the F00942T -> `emdbsrcflg` value is `1`, the F00942T-> `emenvfu2` value is `0` or blank, and the foundation selected for the package is 64-bit, then it sets F00942T -> `emenvfu2 =2` for processing. It does a fetch from the F9860 table for all objects that have F9860-> `sigtffu=1` and that are not APPL, GT, or DSTR, and not system code 88. For each fetch, it retrieves the par file from the F98780R table, expands the par file, converts the include and source files to 64-bit and places them in temporary include64

and source64 directories. It creates the par file in all four directories and inserts it into the repository. When the process is complete, the system sets the value F00942T-> `emenvfu2 =1` to indicate that the files are converted to 64-bit.

9. Client: Initiates the connection with the Primary Enterprise Server.
10. Enterprise Server: Creates the directories on the Primary Enterprise Server and builds the list of objects from the F96225 table.
11. Enterprise Server: If client populates the F98780R/F98780H tables, then the Enterprise Server checks the '*Live*' specs F98710 <package name> for the object names F98780R, F98780H, and F00942T. If these specs are not there, then it get the specs from the Central Objects path code and inserts them into the '*Live*' specs tables, F98710, F98711, F98712, and F98713.
12. Client: Opens the F96511 table to get active package for the primary server using port, server, and pathcode.
13. Client: Opens the F96215 table of the active package and gets the last date and time stamp of the active package.
14. Client: Opens the F96225 table of active package and gets the date and time, and `pkgOper =04`. For all selected records, the system inserts date, time, `pkgOper=04`, and object name into the F96225 table of the new package.
15. Client: Opens the F9861 table and queries on all records with date greater than or equal to the date the active package was built. For each record, the system inserts object into the F96225 table with new package, date, time, `pkgOper=04`, and object name. These are all the changed objects.
16. Enterprise Server: Initiates the client build.
17. Deployment Server: Initializes the client build using the Deployment Server Services.
18. Deployment Server: Sets the Net Change Objects mode for the client build.
19. Deployment Server: Initiates the business services process.
20. Deployment Server: Starts the copy of `bin64, lib64, obj, source, include, source64, include64, res`, and `work` from the active package to this current package.
21. Deployment Server: Waits for the Enterprise Server to complete the Enterprise Server copy of the files.
22. Enterprise Server: Builds the object list of the records in the F96225 table.
23. Enterprise Server: UNIX/Windows: Starts the Builds specs on a thread (IBM i will not be on a thread) on the Primary Enterprise Server from Central Objects, placing the results in the spec tables of the package, <table name>_<package name> (for example, F98710_DV920FA) in the database. There are 17 tables that are copied from one location to the other. Also, the manifest table is created which holds the name of the package. This is running on a thread so the package can continue.
24. Enterprise Server: Copies the bin64, lib64, include, source, and obj from the active package to the new package.
25. Enterprise Server: Opens F96225 for package name where `pkgOper=04`. For each object, the system retrieves the files from the repository, validates each BSFN and TBLE that there is a record in the F98780R table, the record has a par file, and the `.h` and/or `.c` file exist in the par file.
26. Enterprise Server: Validates each UBE and BSVW with F9860->`sigtffu1=1` if there is a record in the F98780R table, the record has a par file, and for each UBE and BSVW, if the `.h` file exists in the par file and for an APPL, if a bitmap exists.
27. Enterprise Server: If the validation fails and the Repair Repository flag is set, then the system inserts a record into the F96225 table with package name, pathcode, primary server name, object name, version = "`REPAIR`", operation = "`11`", and date and time.
28. Enterprise Server: Checks if any BSFN or TBLE does not have F9860->`sigtffu1=1` and is checked in. If found, the system inserts a record into the F96225 table with package name, pathcode, primary server name, object name, version = "`REPAIR`" and option = "`11`", and date and time.
29. Client: Compresses the **res** directory from the pathcode location on the Deployment Server. The system transfers the `res.cab` file to the Primary Enterprise Server under the `checkin\res` folder.
30. Client: If the Repair Repository flag is set, then the system selects F96225 records where package name, pathcode, primary server, operation="`11`", and status = `ERROR`.

    For each record, the system attempts to get the record from the F98780R table, evaluate what is missing. If the record is not in the F98780R table, the system recreates the par file and inserts the record into the F98780R and F98780H tables.

31. Enterprise Server: If the Repair Repository flag is set, the system selects the F96225 records where package name, pathcode, primary server, operation="`11`", and status = `ERROR`. The system then retrieves and validates each record in the F98780R table if there is a **.h** or/and **.c** file. If successful, then the system sets the F96225 record to status=`SUCCESS`.
32. Enterprise Server: If there are other servers, the Primary Enterprise Server will contact each Enterprise Server to start copying the files from the active package to the new package on that Enterprise Server.
33. Enterprise Server: On the Primary Enterprise Server, the package build checks if there is a difference between data dictionary items in the database compared to what is in **ddict/ddtext**. The difference fields are the type and length. If there are any differences, then it locks all processes to prevent any UBE to run and waits for all UBEs to finish. It does not affect users on the Web. When the locks are obtained, **ddict/ddtext/glbtb** are deleted in the path code spec directory.
34. Enterprise Server: Opens the F96225 table for package name and `pkgOper=04`. If object is named event rules (NERs), then it creates `.c` and `.h` files in the **checkin\source64** and **checkin\include64** directories (if the system on the Enterprise Server is 64-bit).
35. Enterprise server: If the client has finished copying, the Primary Enterprise Server transfers the files from the **checkin\source, checkin\source64, checkin\include, checkin\include64 and checkin\res** to the Deployment Server package directories.
36. Enterprise Server: Moves server-only `.c` and `.h` files in **checkin\source, checkin\source64, checkin\include, checkin\include64 and checkin** directory to the source and include directories respectively on the Primary Enterprise Server. If there is more than one server, then the Primary Enterprise Server moves the `.c` and `.h` files from the source and include directories to the other Enterprise Servers under the source and include directories.
37. Enterprise Server: Compiles business functions on the Primary Enterprise Server to generate `.dll`, `.so`, `.sl`, or `.SRVPGM` files. If there are multiple servers, the Primary Enterprise Server sends a message to the other servers to start building the business functions.
38. Enterprise Server: A new unique library `IBMi` is created in which you can create the SVRPGM, MODULES, and FILES. This unique library is created with the first two letters and the last two letters of the package, and the job number. For example, if the package is DV920FA, then the unique library is created as "`DVFA000340`". This name is stored in the F9603 table in `phpkgdeffut3`.
39. Enterprise Server: Checks the status of the other servers to see if BSFNs are complete. Checks its own status to see if BSFNs are complete.
40. Enterprise Server: When the dll and specs are finished, if the compress feature for server is enabled, the system compresses the `.dll`, `.so`, `.sl`, and `.SRVPGM` files on the Primary Enterprise Servers. The Primary Enterprise Server sends a message to all other servers to compress.
41. Enterprise Server: Transfers the compressed files and the `compress.inf` file back to the Deployment Server under the **<packagename>/<machine type>** directory.
42. UNIX/Windows: Under the deploy location, creates the package name under the bin64 and specs folders in these paths, respectively: **<pathcode>\bin64\<package name>** and **<pathcode>\spec\<package name>**.
43. IBM i: Creates a new library with the package name.
44. IBM i: Copies SVRPGM, MODULES, and FILES from the unique library to the package name library.
45. UNIX/Windows: Copies the package bin64 directory to the **<pathcode> \bin64\<package name>** directory.
46. Under **<pathcode>\spec\<package name>**, creates the `spec.ini` file with the database datasource location for Central Objects.
47. Enterprise Server: Checks the `jde.ini` file for entry.

    ```
    [PACKAGE MANIFEST]
     CreateJAR=Y
    ```

    and/or checks the F96215->`PKGBULFUT3 = "Y"` which is set when you select **Compress Application Component** in the Compress Options tab.
    If this entry is in `jde.ini`file or F96215 table with '`Y`', then the system creates a `E1_APP_xxxx.jar` file under the package name directory on the Enterprise Server. This jar file contains the bin64 directory and the `spec.ini`

populated with the package name. The jar file is used by the Server Manager to create a new Enterprise Server instance.

48. Client machine: Checks client build status to see if the client build is complete. Continues to check the status until the build is complete.
49. Enterprise Server: Transfers the server log (`svrpkgbuild.log`) from each server to the Deployment Server under **<packagename>\serverLogs** directory.
50. Enterprise Server: Transfers the files located under the `text` directory on each Enterprise Server to the Deployment Server under **<packagename>\serverlogs\<server name>\text**.
51. Enterprise Server: Transfers the files located under the `compileLogs` directory on each Enterprise Server to the Deployment Server under **<packagename>\serverlogs\<server name>\CompileLog.**
52. Enterprise Server: Transfers the files located under the `GenerateNER_logs` directory on the Primary Enterprise Server to the Deployment Server under **<packagename>\serverlogs\GenerateNER_logs.**
53. The process is complete. The client build is initiated from the Deployment Server Services running on a thread parallel to the server package build.
54. Deployment Server: Runs the client package initiated from the Deployment Server Services. A message is sent from the Primary server to the Deployment server to start the client build.
55. Deployment Server: Initializes the client package and sets the Net Changed flag.
56. Deployment Server: Creates the object list from the F96225 table.
57. Deployment Server: Creates the package `inf` file.
58. Deployment Server: Gets all the `SBF` files from the F98780R repository table. Builds the BSSV - Business Services on a separate thread.
59. Deployment Server: Copies the files from the active package to the new package. The system copies bin64, lib64, obj, source, include, source64, include64, and rest.
60. Deployment Server: Waits for the Enterprise Server to copy the new Net Changed objects to the Deployment Server.
61. Deployment Server: Compiles the `.c` and `.h` files using Busbuild.
62. Deployment Server: Creates the `spec.ini` file that contains the package name and the Central Objects data source. The `spec.ini` file points to specs of the package tables in the Central Objects data source. The local spec database is no longer created.
63. Deployment Server: Waits for compilation of the business functions and the BSSV build to complete.
64. Deployment Server: Compresses the directories on the Deployment Server if compression is selected.

The Client and Server package build is complete. You can now deploy the server package.

How the System Deploys the Full Server Package

This is an overview of how the JD Edwards EnterpriseOne system deploys a full server package and the affect it has on the JAS Server.

Note: Any references to 64-bit apply to Tools Release 9.2.3 and higher.

Full Package - Normal Package Deployment (Prior to Tools Release 9.2.6)

1. Set the F9651 field mdmchrcdnm=50 to prepare the server for Deploy.
2. Process sleeps for 1 minute which allows the JAS server polling to see there is a package to deploy. The JAS server will put a lock on the web applications to wait for the deploy to finish. The JAS server will poll every 5 seconds checking the F9651 table to see if it is done with the deploy. It looks for mdmchrcdnm=30. There are no effects to user already in the application only if the users try to bring up a new application will they have to wait.
3. Sets the F9651 field mdmchrcdnm=10 to indicate deployment is happening.
4. Enterprise Server sends message to all kernels to lock. This will prevent any UBE from being submitted.

5. Process waits for all running UBEs to complete. It will only wait as long as the setting in the jde.ini indicated - "DeployWaitInMinutes=x" or default to 30 minutes. If time has expired, it will release the locks without deploying the package.
6. Once UBEs are finished running:
    a. Opens the <pathcode>/spec.ini file and changes the <pathcode>_Package setting to the new package name:

       ```
       DV920_Package=DV920FA
       ```
    b. Copies the .dll/.so/sl/SVRPGM from the package directory to the "Live Area" which is <pathcode>\bin32/bin64" or the SVRPGM library.
    c. Deletes the ddict, ddtext and glbtbl specs from the <pathcode>/spec directory.
    d. Deletes the Runtime cache from the <pathcode>/runtimeCache directory.
    e. Releases the kernel locks.
    f. Sets the F9651 field mdmchrcdnm=30 to indicate deployment is finished.
7. Opens the <pathcode>/spec.ini file and changes the <pathcode>_Package setting to the new package name: **DV920_Package=DV920FA**.
8. Sends a broadcast message that a package change was performed.
9. Sets the F9651 mdmchrcdnm field value to 30 to indicate deployment is complete.
10. Updates the record in the F96511 table with Enterprise Server information, port number, pathcode, package name, and type 30 to indicate this is the deployed package.
11. (Starting with Tools Release 9.2.5.0) If the package is also a client package, insert a record in the F98825 table with these details:
    - MKEY=”CLIENT"
    - UPJDEPKGNM=<package name>
    - UPPATHCOD=<pathcode>
    - UPINPKST=’20’
    - Date and time

    For the previously deployed package with the details MKEY=”CLIENT” and UPINPKST=’20’, update the status to UPINPKST=’30.

Full Package - Normal Package Deployment – Zero Downtime (Tools Release 9.2.6)

1. Validates the Enterprise Server.
2. Opens the **<pathcode>/spec.ini** file and changes the `<pathcode>_Package` setting to the new package name: **DV920_Package=DV920FA**.
3. Sends a broadcast message that a package change was performed.
4. Updates the record in the F96511 table with Enterprise Server information, port number, pathcode, package name, and enters `30` in the `mdmchrcdnm` field to indicate this is the deployed package.
5. Sets the F9651 `mdmchrcdnm` field value to `30` to indicate that the deployment is complete.
6. If the package is also a client package, inserts a record in the F98825 table with these details:

    # `MKEY="CLIENT"`

    # `UPJDEPKGNM=<package name>`

    # `UPPATHCOD=<pathcode>`

    # `UPINPKST='20'`

    # `Date and time`

7. For the previously deployed package with the details `MKEY="CLIENT"` and `UPINPKST='20'`, updates the status to `UPINPKST='30'`.

Action Taken on the JAS Server

When the first person signs on to a web application after the full package deployment, the process checks the manifest table, F98770, discovers that a package was deployed, and follows these steps:

1. Sets locks on the web users.
2. Updates the manifest table to the new full package.
3. Releases the locks on the web users enabling them to sign in to the applications.

Action Taken on the JAS Server (Tools Release 9.2.6.0)

When the first person signs on to a web application after the full package deployment, the process checks the manifest table, F98770, discovers that a package was deployed, and follows the step:

1. Updates the manifest table to the new full package. After this step, users can sign in to the applications.

How the System Deploys the Server Update Package

This is an overview of how the JD Edwards EnterpriseOne system deploys different types of update server packages.

Note: Any references to 64-bit apply to Tools Release 9.2.3 and higher.

Deployment of Update Package that Includes All Types of Objects:

1. Sets the F9651 field mdmchrcdnm=50 to prepare the server for deployment.
2. Process sleeps for 1 minute which allows the JAS server polling to see there is a package to deploy. The JAS server puts a lock on the web application to wait for the deployment to finish. The JAS server polls every 5 seconds, checking the F9651 table to see if it is done with the deployment. It looks for mdmchrcdnm=30. There are no effects to users already in the application, only if the users try to bring up a new application.
3. Sets the F9651 field mdmchrcdnm=10 to indicate deployment is happening.
4. Enterprise Server sends message to all kernels to lock. This will prevent any UBEs from being submitted.
5. Process waits for all running UBEs to complete. It will only wait as long as the setting in the jde.ini indicates - "DeployWaitInMinutes=x" or default to 30 minutes. If time has expired, it will release the locks without deploying the package.
6. Once UBEs are finished running:
   a. Merges the update package tables with the current database tables to which the spec.ini points.
   b. Copies the .dll/.so/sl/SVRPGM from the update package directory to the "Live Area": <pathcode>\<packagename>\bin64" or the package name SVRPGM library.
   c. Deletes the ddict, ddtext, and glbtbl specs from the <pathcode>/spec/<package name> directory.
   d. Deletes only the directories of the UBEs that are in the package from the runtime cache directory, <pathcode>/<package name>/runtimeCache.
7. Releases the locks and submits any UBE in the queue.
8. Sets the F9651 field mdmchrcdnm= 30 to indicate deployment is finished.
9. (Starting with Tools Release 9.2.5.0) If the package is a client package, on the Deployment Server, copy the bin32 ,(64) obj, lib32(64), include, include64, source and source64 to the parent package.

Deployment of an Update Package that Contains Only UBE Objects:

1. Sets the F9651 field mdmchrcdnm=50 to prepare the server for deployment.

2. Sets the F9651 field mdmchrcdnm=10 to indicate deployment is happening.
3. Places a lock on only the UBEs in the package. If there is a submit to run a UBE in the package, it is inserted into the F988259 table and held until the package is complete.
4. Merges the update package tables with the current database to which the spec.ini points.
5. Deletes only the directories of the UBEs that are in the package from the Runtime cache directory, <pathcode>/<package name>/runtimeCache.
6. (Starting with Tools Release 9.2.5.0) If the package is a client package, on the deployment server, include, include64, source and source64 to the parent package.
7. Sets the F9651 field mdmchrcdnm= 30 to indicate deployment is finished.
8. Submits the list of UBEs in the F988259 to run.

Note: Only the UBEs in the package are not allowed to run. All other UBEs will be processed during deployment.

Deployment of an Update Package that Contains Only Application Objects:

1. Sets the F9651 field mdmchrcdnm=50 to prepare the server for deployment.
2. The process sleeps for one minute, which allows the JAS Server polling to see there is a package to deploy. The JAS Server will put a lock on applications to wait for deployment to finish. It will start polling every 5 seconds checking the F9651 table to see if it is done with deployment. It looks for mdmchrcdnm=30. There is no effect to users already in the application. If starting an application, there will be a wait until deployment is finished.
3. Sets the F9651 field mdmchrcdnm=10 to indicate deployment is happening.
4. Merges the update package tables with the current database tables to which the spec.ini points.
5. Sets the F9651 field mdmchrcdnm=30 to indicate deployment is finished.

Deployment of an Update Package that Contains Only UBE and Application Objects:

1. Set sthe F9651 field mdmchrcdnm=50 to prepare the server for deployment.
2. The process sleeps for one minute, which allows the JAS Server polling to see that there is a package to deploy. The JAS Server puts a lock on applications to wait for deployment to finish. It starts polling every 5 seconds checking the F9651 table to see if it is done with deployment. It looks for mdmchrcdnm=30. There is no effect to users already in the application. If starting an application, there will be a wait until deployment finishes.
3. Sets the F9651 field mdmchrcdnm=10 to indicate deployment is happening.
4. Places a lock on only the UBEs in the package. If there is a submit to run a UBE in the package, it is inserted into the F988259 table and held until the package is complete.
5. Merges the update package tables with the current database tables to which the spec.ini points.
6. Releases the locks and submits any UBE in the queue.
7. Sets the F9651 field mdmchrcdnm=30 to indicate deployment is finished.
8. (Starting with Tools Release 9.2.5.0) If the package is a client package, on the deployment server, copy, include, include64, source and source64 to the parent package.
9. Submits the list of UBEs in the F988259 to run.

Note: Only the UBEs in the package are not allowed to run. All other UBEs are processed during deployment.

How the System Builds Packages from the Web (Release 9.2.5)

This section provides an overview of how the JD Edwards Enterprise One system builds a full server and a client package from the web. Starting with Tools Release 9.2.5.0, you can only build a server-only package or a server and client package. Building client-only packages is not supported.

Starting with Tools Release 9.2.5.0, a new JDENET service is running on the deployment server to accept messages from the enterprise server. The enterprise server can send messages to the deployment server to build the package on the client. You can start and stop this service during the installation of the Tools Release from the Server Manager.

Note: Ensure that the port number on the deployment server matches the port number of the enterprise server that is building the package.

How the System Builds a Full Server Client Package from the Web

1. Web: Server Package UBE (R9621S) initiates the server package build.
2. Web: UBE (R9621S) calls the BSFN program B9600196 on the enterprise server which calls into the package kernel code.
3. Web Enterprise Server: The system gets the primary server name and sends a jdenet message to the primary server to start the package.
4. Web Enterprise Server: The system does not wait for a response but goes into a continuous loop checking the status of the build, F96021-> bhbldsts, until it is finished.
5. Primary Server: The primary server receives the message to start the package build.
6. Primary Server: The system creates the package build directories on the primary enterprise server.
7. Primary Server: The system checks the F00942T->emdbsrcflg value for 1. If value is 0 or blank, the objects have not been inserted into the F98780R table. If value is 1, then the objects have been inserted.
8. Primary Server: The system checks the F00942T-emenvfu2 value for 1. If the value is 0 or blank, then the include and source files in the F98780R/R98780H have not been converted to 64-bit. If the value is 1, then the include and source files have been converted to include64 and source64 and uploaded into the F98780R/R98780H tables.
9. Primary Server: If the files need to be inserted into the repository, send a jdenet message to the deployment server to initialize.
10. Deployment Server: Initializes the deployment server and create the work directory.
11. Primary Server: Sends a message to the deployment server to insert the records into the repository and waits for it to finish. This can take more than four hours.
12. Deployment Server: Receives the message from the primary server to insert records into the repository.
13. Deployment Server: Inserts all the records into the repository once. If F00942T -> emdbsrcflg value is 0 or blank and the F00942T-> emenvfu2 value is 0 or blank, then the system sets the F00942T -> emdbsrcflg=2 for processing. It does a fetch from F9860 for all objects that are not system code 88. It checks if there is a file associated with the object on the deployment server pathcode. If there are files then it creates a par file for the object and insert the record with the par file into the F98780R/F98780H. If the foundation selected for the package was 32-bit, then for business functions and tables, the par file will only include source and include files. It sets the object record in the F9860->sigtffu =1 to indicate there are files for this object. When the process is complete, it sets the F00942T->embdsrcflg=1 for completion. It leaves the F00942T-> emenvfu2 as 0 or blank since this indicates that the conversion of the include and source into 64-bit files was not done. If the foundation selected for the package was 64-bit, then for business functions and tables, it converts the include and source into 64-bit files into a temporary include64 and source64 directory. It creates the par file with all four directories and inserts the record into the F98780R/F98780H. It sets the object record in the F9860->sigtffu =1 to indicate there are files for this object. When the process is complete, it sets the F00942T->embdsrcflg=1 for completion. It also sets the F00942T-> emenvfu2 =1, which indicates the conversion of the include and source into 64-bit files was done.
14. Deployment Server (Converts 32-bit to 64-bit): This is only done once. If F00942T -> emdbsrcflg value is 1, the F00942T-> emenvfu2 value is 0 or blank, and the foundation selected for the package was 64-bit, then it sets F00942T -> emenvfu2 =2 for processing. It does a fetch from F9860 for all objects that have F9860->sigtffu=1 and that are not APPL, GT or DSTR, and not system code 88. For each fetch, it retrieves the par file from the F98780R, expands the par file, converts the include and source files to 64-bit and places them in temporary include64 and source64 directories. It creates the par file with all four directories and inserts it into

the repository. When complete, it sets the F00942T-> emenvfu2 =1 to indicate the files have been converted to 64-bit.

15. Deployment Server: When finished, returns to the Primary Server.
16. Primary Enterprise Server: Sends a message to the deployment server to initialize deployment server.
17. Deployment Server: Creates the directories.
18. Primary Enterprise Server: Builds the list of objects from the F96225 table.
19. Primary Enterprise Server: If client populated the F98780R/F98780H, then the system checks the specs F98710 (package name) for the object name F98780R. If these specs are not there, then it get the specs from the central objects path code and insert them into the live specs tables, F98710, F98711, F98712 and F98713.
20. Primary Enterprise Server: If F00942T -> emdbsrcflg= 1, opens the F9860 and gets all objects where a F9860->sigtffu1=1 and F9860->sy != 88 and not a Named Event Rule (NER) or a Business Service (SBF) object. Open the F98780R and retrieve each object from F98780R which will put the files into primary enterprise server directories, checkin\source (.c files), checkin\source64 (.c files), checkin\include(.h files) , checkin\include64(.h files) and the bitmaps into the checkin\res directory.
21. Enterprise Server: (Release 9.2.7.3) The system validates each BSFN and TBLE if there is a record in the F98780R table, the record has a par file and if the .h and/or .c file exist in the checkin location.
22. Enterprise Server: (Release 9.2.7.3) The system validates each UBE, BSVW, and APPL with F9860->sigtffu1=1 if there is a record in the F98780R table, the record has a par file and for a UBE and BSVW, if the .h file exists in the checkin location and for an APPL, if a bitmap exists.
23. Enterprise Server: (Release 9.2.7.3) If the validate failed and the Repair Repository flag is set, then the system inserts a record into the F96225 table with package name, pathcode, primary server name, object name, version = “REPAIR”, and operation = “11”.
24. Enterprise Server: (Release 9.2.7.3) The system checks if any BSFN or TBLE does not have F9860->sigtffu1=1 and is checked in. If found, the system inserts a record into the F96225 table with package name, pathcode, primary server name, object name, version = “REPAIR”, and option = “11”.
25. Enterprise Server: (Release 9.2.7.3) If the Repair Repository flag is set and there are REPAIR records inserted in F96225, then send message to Deployment Server Services to Repair the objects.
26. Deployment Server: (Release 9.2.7.3) If the Repair Repository flag is set then the system selects the F96225 records where package name, pathcode, primary server, operation=”11” and status = ERROR. For each record, the system attempts to get the record from the F98780R table, evaluate what is missing and if it is not in the F98780R table, recreate the par file and insert the record back into the F98780R and F98780H tables.
27. Enterprise Server: (Release 9.2.7.3) If the Repair Repository flag is set then the system selects the F96225 records where package name, pathcode, primary server, operation=”11”, and status = ERROR. It then retrieves and validates each record from the F98780R table if there is a .h or/and .c file. If the validation is successful, the system sets the F96225 record to status=SUCCESS.
28. Primary Enterprise Server: If there are other servers, the primary enterprise server will contact each of the enterprise servers to start the build process.
29. Primary Enterprise Server: On the primary enterprise server, the package build checks if there is a difference between the data dictionary items in the database and the data dictionary items in ddict/ddtext. Differences can exist between the type and length fields in the two locations. If there are any differences, then it locks all processes to prevent any UBE to run and waits for all UBEs to finish. It does not affect users on the web. When the locks are obtained, ddict/ddtext/glbtb are deleted in the path code spec directory.
30. Primary Enterprise Server: The Named Event Rules (NERs) are generated in the checkin\source and checkin \include directories if the system on the Enterprise Server is 32-bit. If the system on the enterprise server is 64-bit, then it will generated the files into the checkin\source64 and checkin\include64 on the primary enterprise server.
31. Primary Enterprise Server: Moves client or server and server-only .c and .h files (if 32-bit system moves the checkin include and source, if 64-bit system moves the include64 and source64) to the source and include directories respectively on the primary enterprise server.
32. Primary Enterprise Server: If there are more than one server, then the primary enterprise server moves the .c and .h files from the source and include directories to other enterprise servers under the source and include directories.

33. Primary Enterprise Server: Compiles business functions on the primary enterprise server to generate .dll, .so, .sl, or .SRVPGM files.
34. Primary Enterprise Server: IBM i (Starting with Tools Release 9.2.6.0) A new unique library is created in which you can create the SVRPGM, MODULES and FILES. This unique library name is created with the first 2 letters and the last 2 letters of the package, and the job number. For example if the package is DV920FA, then the unique library is created as "DVFA000340". This name is stored in the F9603 file in phpkgdeffut3.
35. Primary Enterprise Server: If there are multiple servers, the primary enterprise server sends a message to the other servers to start the build of business functions.
36. Primary Enterprise Server: Builds specs on the primary enterprise server from central objects, placing the results in the package's spec tables, <table name>_<package name> (for example, F98710_DV920FA) in the database. There are 17 tables that are copied from one location to the other. Also, the manifest table is created which holds the name of the package.
37. Primary Enterprise Server: The Primary Enterprise Server creates the tables F989999 and F989999 in Central Objects location with the name F989999_<package name> and F999999_<package name>.
38. Primary Enterprise Server: If the client package was also selected, the primary enterprise server transfers the files from the checkin\source, checkin\source64, checkin\include, checkin\include64 and checkin\res to the deployment server package include, source, include64, source64 and res directories.
39. Primary Enterprise Server: Checks the status of the build and checks if BSFNs are complete. THe primary enterprise server checks the status of other servers if BSFNs are complete.
40. Primary Enterprise Server: When the dll and specs are finished, if the compress feature for server is enabled, it compresses the .dll, .so, .sl, and .SRVPGM files on the primary enterprise servers. The primary enterprise server sends a message to all other servers to compress.
41. Other Ent Server: Sends compress files to the primary ent server under <packagename>/<machine type> directory.
42. Primary Enterprise Server: Transfers the compressed files and the compress.inf file back to the deployment server if initialized was successful under the <packagename>/<machine type> directory.
43. (Starting with Tools Release 9.2.6.0) Under the deployment location, creates the package name under the bin64 and specs folder in these paths, respectively: <pathcode>\bin64\<package name> and <pathcode>\spec \<package name>.
44. (Starting with Tools Release 9.2.6.0) Copies the package bin64 directory to the <pathcode>\bin64\<package name> directory.
45. (Starting with Tools Release 9.2.6.0) Under <pathcode>\spec\<package name>, creates the spec.ini file with the database datasource location for Central Objects

    DV920_DataSource=Central Objects – DV920
46. Primary Enterprise Server: Check the jde.ini for entry

    ```
    [PACKAGE MANIFEST] CreateJAR=Y
    ```

    (Release 9.2.7.0) and/or check the F96215->PKGBULFUT3 = "Y" which is set when you select **Compress Application Component** in the compress options tab.

    If this entry is in jde.ini or F96215 with 'Y', then create a E1_APP_xxxx.jar file under the package name directory on the Enterprise Server. This jar file contains the bin64 directory and the spec.ini populated with the package name. The jar file is used by the Server Manager to create a new Enterprise Server Instance.
47. Primary Enterprise Server: Transfers the server log (svrpkgbuild.log) from each server to the deployment server under 38. <packagename>\serverLogs directory.
48. Primary Enterprise Server: Transfers the files located under the text directory on the each enterprise server to the deployment server under <packagename>\serverlogs\<server name>\text.
49. Primary Enterprise Server: Transfers the files located under the CompileLogs directory on the each enterprise server to the deployment server under (packagename>\serverlogs\(server name>\CompileLog.
50. Primary Enterprise Server: Transfers the files located under the GenerateNER_logs directory on the primary enterprise Server to the deployment server under (packagename>\serverlogs\GenerateNER_logs.

51. If this is a server only package, the process is complete. However, if client was selected, the system performs the next steps.
52. Web R9621S UBE: Runs Client Package UBE R9622C to initiate the client package build.
53. R9622C call B9600186 for client package build.
54. Web Enterprise server: Creates and sends a message to the deployment server to start the build.
55. Web enterprise server: Waits for the build to finish.
56. Deployment Server: Creates the package inf file.
57. Deployment Server: Compiles the .c and .h files using Busbuild.48.
58. Deployment Server (Startin with Tools Release 9.2.5.0): Creates the spec.ini with the package name and central object datasource. The spec.ini points to the package tables specs in the central objects datasoure. The local spec database is no longer created.
59. Deployment Server (Prior to Tools Release 9.2.5.0): Builds the specs from the package's spec tables (table name>_(package name> (for example, F98710_DV920FA) in the database and puts them into the package's local spec database.
60. Deployment Server: If F00942T-> emdbsrcflg= 1 or 3 then gets all the SBF files from the F98780R repository table. If the field was blank or 0 then copies the files from (pathcode>\sbf directory. Build the BSSV - Business Services.
61. Deployment Server: Waits for the compile of the business functions, building of the specs, and the building of the BSSV to complete.
62. Deployment Server: Compresses the directories on the deployment server if compression was selected.
63. Deployment Server: Sends a message back to enterprise server that client package is complete.
64. Web enterprise Server: Receives a message and completes processing both R9621S and R9622C.

How the System Builds an Update Server Client Package from the Web

This section provides an overview of how the JD Edwards EnterpriseOne system builds an update server and client package from the web. The beginning of each step states whether the process occurs on the client from where the build is performed or on the Deployment server, Primary Enterprise Server or Web Enterprise server.

Note: Any references to 64-bit apply to Tools Release 9.2.3 and higher.

1. Web: Server Package UBE (R9621S) initiates the server package build.
2. Web: UBE (R9621S) calls the bsfn B9600196 on the enterprise server which calls into the package kernel code.
3. Web Enterprise Server: The system gets the primary server name and sends a jdenet message to the primary server to start the package.
4. Web Enterprise Server: The system does not wait for a response but goes into a continuous loop checking the status of the build, F96021-> bhbldsts, until it is finished.
5. Primary Enterprise Server: The system initializes the primary enterprise server.
6. Primary Enterprise Server: Creates directories on the primary enterprise server.
7. Primary Enterprise Server: Sends a message to the deployment server to initialize the deployment server.
8. Deployment Server: Creates the directories on the deployment server.
9. Primary Enterprise Server: The system builds a link list of objects in the update package from the F96225 table. It builds the list of business functions, NER, and Table ER in a file called "bsfnfile.txt". A list of bitmaps are listed in a file called "bitmapfile.txt". Both files are transferred from the <package name>\work directory to the deployment server.
10. Primary Enterprise Server: If F00942T->emdbsrcflg=1, then the system opens the link list of objects and if the F9860->sigtffu1, it retrieves the par file from the F98780R and puts them in the checkin\include, checkin \include64, checkin\source, checkin\source64 and checkin\res directories.
11. Enterprise Server: (Release 9.2.7.3) For each object, the system validates each BSFN and TBLE if there is a record in the F98780R table, the record has a par file, and if the .h and/or .c file exist in the checkin location.

12. Enterprise Server: (Release 9.2.7.3) The system validates each UBE, BSVW, and APPL with F9860->sigtffu1=1 if there is a record in the F98780R table, the record has a par file and for a UBE and BSVW, if a .h file exists in the checkin location and for an APPL, if a bitmap exists.
13. Enterprise Server: (Release 9.2.7.3) If the validation fails and the Repair Repository flag is set, then the system inserts a record into the F96225 table with the package name, pathcode, primary server name, object name, version = “REPAIR”, operation = “11”, and date and time.
14. Enterprise Server: (Release 9.2.7.3) For each UBE, BSVW, and APPL which does not have F9860->sigffu1=1, the system inserts a record into the F96225 table as a REPAIR record. The client checks if there is a file associated with it and repairs it.
15. Enterprise Server: (Release 9.2.7.3: If the REPAIR Repository flag is set and there are records in the F96225 table that need to be repaired, the ssytem sends a message to the deployment server to repair the records.
16. Deployment Server: (Release 9.2.7.3) If the Repair Repository flag is set, the system selects F96225 records where package name, pathcode, primary server, operation=”11”, and status = ERROR. For each record retrieved from the F98780R table, the system evaluates what is wrong or missing and recreates the par file. The system then inserts the record back into the F98780R and F98780H tables.
17. Enterprise Server: (Release 9.2.7.3) If the Repair Repository flag is set, the system selects F96225 records where package name, pathcode, primary server, operation=”11” and status = ERROR. It retrieves and validates each record from the F98780R table if there is a .h or/and .c file or if there is a bitmap. If the validation is successful, the system sets the F96225 record to status=SUCCESS.
18. Primary Enterprise Server: On the primary enterprise server, if the objects in the update package contain NER, on the primary enterprise server, the package build checks if there is a difference between data dictionary items in the database and the data dictionary items in ddict/ddtext. Differences can exist between the type and length fields in the two locations. If any differences exist, then the package build locks all the processes to prevent any UBE to run and waits for all UBEs to finish. It does not affect users on the Web. When the locks are obtained, ddict/ddtext/glbtb are deleted in the path code spec directory. The locks are then released.
19. Primary Enterprise Server: The system generates named event rules (NERs) on the primary enterprise server. The generated files are created either in the checkin\source and checkin\include directories or checkin \source64 and checkin\include64 directories depending on the bitness.
20. Primary Enterprise Server: The primary enterprise server copies server only .c and .h files to the source and include directories respectively. If multiple servers are available, then the primary enterprise server transfers the .c and .h files to the other enterprise servers under the source and include directories.
21. Primary Enterprise Server: If 64-bit is enabled in the release master, only the include64 and source64 artifacts are copied into the checkin location (Starting with Tools Release 9.2.4)
22. Primary Enterprise Server: Copies the parent dlls from the parent package to the update package. This dll is used in the compile of the business functions for the update package. If there are multiple servers, the primary enterprise server sends a message to the other servers to start the build of business functions.
23. (Starting with Tools Release 9.2.6.0) IBM i: Opens the F9603 to get the unique library of the parent package from phpkgdeffut3. It copies SVRPGM to the update package library, and compiles the business functions and links with the library location of the parent package.
24. Primary Enterprise Server: The primary enterprise server builds specs from central objects, placing the results in the package's spec tables, <table name>_<package name> (for example, F98710_DV920FA) in the database. The 17 tables are created and the records for the objects are inserted into the new tables. The manifest table is created with the name of the package and all the objects in the update package.
25. Primary Enterprise Server: If the client package was selected, the primary enterprise server transfers the files from the checkin\source, checkin\include, checkin\source64 and checkin\include64 to the deployment server package include, source, include64 and source64 directories.
26. Primary Enterprise Server: If client package was selected, the primary enterprise server transfers back res bitmaps from the checkin\res to the deployment server in the <package name>\res directory.
27. Primary Enterprise Server: The system waits for specs to finish and for dlls to finish building. If there were other servers, it will wait for them to complete.
28. Primary Ent Server: When the dlls and specs are finished building, if the compression was selected for the server, then the system compresses the .dll, .so, .sl, and .SRVPGM files on the enterprise servers.

29. Primary Enterprise Server: The primary enterprise server transfers the compressed files and the compress.inf file back to the deployment server under the <packagename>/<machine type> directory.
30. Primary Enterprise Server: The primary enterprise server transfers the server log (svrpkgbuild.log) from each server to the Deployment Server under <packagename>\serverLogs directory.
31. Primary Enterprise Server: Transfers the files located under the text directory on each enterprise server to the deployment server under <packagename>\serverlogs\<server name>\text.
32. Primary Enterprise Server: If 64-bit is enabled in Release Master, only the include64 and source64 artifacts are copied to the client and the res files to the deployment server (Starting with Tools Release 9.2.4.0).
33. Primary Enterprise Server: Transfers the files located under the CompileLogs directory on each enterprise server to the deployment server under <packagename>\serverlogs\<server name>\CompileLogs.
34. Primary Enterprise Server: Transfers the files located under the GenerateNER_logs directory on the Primary Enterprise Server to the Deployment Server under packagename>\serverlogs\GenerateNER_logs.
    If this is a server only package, the process is complete. However, if you have selected to build a client, the system also performs the next steps.
35. Web R9621S UBE: Runs Client Package UBE R9622C to initiate the client package build.
36. R9622C call B9600186 for client package build.
37. Web Enterprise server: Create message and send message to deployment server to start the build.
38. Web enterprise server: Wait for the build to finish
39. Deployment Server: The system creates the package inf file.
40. Deployment Server: The system compiles the .c and .h files using Busbuild. Busbuild copies the dll from the parent package to build the update business functions into the parent package dll.
41. Deployment Server (Starting with Tools Release 9.2.5.0): No specs are built. For each object a .par file is created under the spec\"package name" directory.\
42. Deployment Server (Prior to Tools Release 9.2.5.0): The system builds the specs from the package's spec tables <table name>_<package name> (for example, F98710_DV920FA) into TAM files located under the spec directory.
43. Deployment Server: The system waits for the business functions to compile and the spec build to complete.
44. Deployment Server (optional): The system compresses the directories on the Deployment Server. This is only necessary if building an ESU. The client install process does not use the compressed update package files.
45. Deployment Server: Send message back to Enterprise server that client package is complete.
46. Web Enterprise Server:This server receives the message and completes processing both R9621S and R9622C.

How the System Builds Full Server and Client Package from the Web with Net Change Objects (Release 9.2.9)

This section provides an overview of how the JD Edwards EnterpriseOne system builds a full server and client package from the web with net change objects.

1. Web: Server Package UBE (R9621S) initiates the server package build.
2. Web: UBE (R9621S) calls the BSFN program B9600196 on the Enterprise Server which calls into the package kernel code.
3. Web Enterprise Server: The system gets the primary server name and sends a jdenet message to the primary server to start the package.
4. Web Enterprise Server: The system does not wait for a response but goes into a continuous loop checking the status of the build, F96021-> `bhbldsts`, until it is finished.
5. Primary Server: Receives the message to start the package build.
6. Primary Server: Initializes the Primary Server.
7. Checks if the Net Changed Objects Mode. Opens F9603 and checks F9603->`phpkg4=1`. If set to `1`, sets flag to Net Changed Objects Mode.
8. Primary Server: Creates the package build directories on the Primary Enterprise server.
9. Primary Enterprise Server: If client populated the F98780R/F98780H, then the system checks the specs F98710 (package name) for the object name F98780R. If these specs are not there, then the system gets the specs from the central objects path code and inserts them into the live specs tables, F98710, F98711, F98712, and F98713.

10. Primary Server: Checks the F00942T->`emdbsrcflg` value for `1`. If the value is `0` or blank, the objects have not been inserted into the F98780R table. If the value is `1`, then the objects have been inserted.
11. Primary Server: Checks the F00942T-`emenvfu2` value for `1`. If the value is `0` or blank, then the include and source files in the F98780R/R98780H have not been converted to 64-bit. If the value is `1`, then the `include` and `source` files have been converted to `include64` and `source64` and uploaded into the F98780R/ R98780H tables.
12. Primary Server: If the files need to be inserted into the repository, the system sends a jdenet message to the Deployment Server to initialize.
13. Deployment Server: Initializes the Deployment Server and creates the package directories.
14. Primary Server: Sends a message to the Deployment Server to insert the records into the repository and waits for it to finish.
15. Deployment Server: Receives the message from the primary server to insert records into the repository.
16. Deployment Server: Inserts all the records into the repository once. If F00942T -> `emdbsrcflg` value is `0` or blank and the F00942T-> `emenvfu2` value is `0` or blank, then the system sets the F00942T -> `emdbsrcflg=2` for processing.

    The system does a fetch from F9860 for all objects that are not system code 88. The system checks if there is a file associated with the object on the Deployment Server pathcode. If there are files then the system creates a par file for the object and insert the record with the par file into the F98780R/F98780H. If the foundation selected for the package is 32-bit, then for business functions and tables, the par file will only include the `source` and `include` files. The system sets the object record in the F9860->`sigtffu =1` to indicate there are files for this object.

    When the process is complete, the system sets the F00942T->`embdsrcflg=1` for completion. It leaves the F00942T-> `emenvfu2` as `0` or blank as this indicates that the conversion of the `include` and `source` files into 64-bit files is not done. If the foundation selected for the package is 64-bit, then for business functions and tables, the system converts the include and source into 64-bit files and places them in temporary `include64` and `source64` directories. The system creates the par file with all four directories and inserts the record into the F98780R/ F98780H.

    The system sets the object record in the F9860->`sigtffu=1` to indicate there are files for this object. When the process is complete, the system sets the F00942T>`embdsrcflg=1` for completion. It also sets F00942T-> `emenvfu2 =1`, which indicates that the conversion of the include and source files into 64-bit files is done.
17. Deployment Server (Converts 32-bit to 64-bit): This is a one-time process. If F00942T -> `emdbsrcflg` value is `1`, the F00942T-> `emenvfu2` value is `0` or blank, and the foundation selected for the package is 64-bit, then the system sets F00942T -> `emenvfu2 =2` for processing. The system does a fetch from F9860 for all objects that have F9860-> `sigtffu=1` and that are not APPL, GT, or DSTR, and not system code 88. For each fetch, the system retrieves the par file from the F98780R, expands the par file, converts the include and source files to 64-bit and places them in temporary `include64` and `source64` directories. The system creates the par file with all four directories and inserts it into the repository. When complete, the system sets the F00942T-> `emenvfu2 =1` to indicate the files are converted to 64-bit.
18. Deployment Server: When finished, returns to the Primary Server.
19. Primary Enterprise Server: Initializes all other servers. Sends message to each server to initialize.
20. Primary Enterprise Server: Opens the F96511 to get active package for the primary server using port, server, and pathcode.
21. Primary Enterprise Server: Opens F96215 and from active package, gets the last date and time stamp of the active package.
22. Primary Enterprise Server: Opens F96225 of active package, date, time, and `pkgOper=04`. For all the selected records with error, the system inserts objects into F96225 of the new package: date, time, `pkgOper=04`, object name, and status of 01/not built.
23. Primary Enterprise Server: Opens F9861 and queries on all records with date greater than or equal to the date the active package was built. For each record which is business function, table, UBE with `.h` file, and business view with `.h`, the system inserts object into F96225 with new package: date, time, `pkgOper=04`, object name, and status of 01/not built. These are all the net changed objects.

24. Primary Enterprise Server: Sends a message to the Deployment Server to initialize the Deployment Server.
25. Deployment Server: Sets Net Change Objects mode flag.
26. Deployment Server: Retrieves name of active package and creates the directories on the Deployment Server if not already created.
27. Deployment Server: Builds the list of objects from the F96225 table.
28. Deployment Server: If Business Services is selected, the system retrieves the Business Service files from the repository.
29. Deployment Server: If Business Services is selected, the system starts the Business Service thread.
30. Deployment Server: Starts to copy the files from the active package to the new package: `bin64, lib64, obj, include, source, include64, source 64`, and `work` directories.
31. Primary Enterprise Server: Builds the list of objects from the F96225 table.
32. Primary Enterprise Server: Windows/UNIX builds specs on a separate thread (IBM i will not build on thread) on the Primary Enterprise Server from central objects, placing the results in the package's spec tables, `<table name>_<package name>` (for example, `F98710_DV920FA`) in the database. There are 17 tables that are copied from one location to the other. Also, the manifest table is created which holds the name of the package. The Primary Enterprise Server creates the tables F989999 and F989999 in Central Objects location with the name `F989999_<package name>` and `F999999_<package name>`.
33. Primary Enterprise Server: Sends message to other servers to begin the copy of the files from the active package to the new package. The files to copy are `bin64`, `lib64`, `object`, `include`, and `source`.
34. Primary Enterprise Server: Starts to copy the files from the active package to the new packages. The files to copy are `bin64`, `lib64`, `object`, `include`, and `source`.
35. Primary Enterprise Server: From the list of objects in F96225, retrieves the files for the business function, table, UBE with files, and business view with file.

    The system opens the F98780R and retrieves each F96225 object from F98780R which inserts the files into the Primary Enterprise Server directories, `checkin\source` (`.c` files), `checkin\source64` (`.c` files), (`.h` files), and `checkin\include64` (`.h` files).

    The system validates each BSFN, TBLE, UBE, and BSVW if there is a record in the F98780R table, the record has a par file and if the `.h`, and/or `.c` file exist in `checkin\source` (`.c` files), `checkin\source64` (`.c` files), `checkin\include` (`.h` files), and `checkin\include64` (`.h` files).
36. Primary Enterprise Server: If the validation fails and the Repair Repository flag is set, then the system inserts a record into the F96225 table with package name, pathcode, primary server name, object name, version = "`REPAIR`", and operation = "`11`".
37. Primary Enterprise Server: If the Repair Repository flag is set and there are REPAIR records inserted in F96225, then the system sends a message to the Deployment Server Services to repair the objects.
38. Deployment Server: If the Repair Repository flag is set, then the system selects the F96225 records where package name, pathcode, primary server, operation="`11`" and status = `ERROR`. For each record, the system attempts to fetch the record from the F98780R table, evaluate what is missing and if the record is not in the F98780R table, recreates the par file and inserts the record back into the F98780R and F98780H tables.
39. Primary Enterprise Server: If the Repair Repository flag is set, then the system selects the F96225 records where package name, pathcode, primary server, operation="`11`", and status = `ERROR`. The system then retrieves and validates each record from the F98780R table if there is a `.h` or/and `.c` file. If the validation is successful, the system sets the F96225 record to status=`SUCCESS`.
40. Primary Enterprise Server: On the Primary Enterprise Server, the package build checks if there is a difference between the data dictionary items in the database and the data dictionary items in `ddict/ddtext`. Differences can exist between the type and length fields in the two locations. If there are any differences, then the system locks all the processes to prevent any UBE to run and waits for all UBEs to finish. It does not affect users on the web. When the locks are obtained, `ddict/ddtext/glbtb` are deleted in the path code spec directory.
41. Primary Enterprise Server: Generate NER: Only the Named Event Rules (NERs) in the list of the F96225 are generated in the `checkin\source64` and `checkin\include64`.
42. Primary Enterprise Server: Waits for the client build to finish copying the client active files to the new files.

43. Primary Enterprise Server: Moves `checkin\include64`, `checkin\source64`, `checkin\include`, and `checkin\source` to the Deployment Server.
44. Deployment Server: Starts busbuild to build the business functions on the Deployment Server.
45. Primary Enterprise Server: Moves client or server and server-only `.c` and `.h` files (if 32-bit system moves the `checkin\include` and `checkin\source`, if 64-bit system moves the `checkin\include64` and `checkin\source64`) to the `source` and `include` directories respectively on the Primary Enterprise Server.
46. Primary Enterprise Server: If there is more than one server, then the Primary Enterprise Server moves the `.c` and `.h` files from the source and include directories to other Enterprise Servers under the `source` and `include` directories.
47. Primary Enterprise Server: Compiles business functions on the Primary Enterprise Server to generate `.dll`, `.so`, `.sl`, or `.SRVPGM` files.
48. Primary Enterprise Server: A new unique library `IBMi` is created in which you can create the SVRPGM, MODULES, and FILES. This unique library name is created with the first two letters and the last two letters of the package, and the job number. For example, if the package is DV920FA, then the unique library is created as "`DVFA000340`". This name is stored in the F9603 file in `phpkgdeffut3`.
49. Primary Enterprise Server: If there are multiple servers, the Primary Enterprise Server sends a message to the other servers to start the build of business functions.
50. Primary Enterprise Server: If the client package was also selected, the Primary Enterprise Server transfers the files from the `checkin\source, checkin\source64, checkin\include, checkin\include64` and `checkin\res` to the Deployment Server Package `include`, `source`, `include64`, `source64`, and `res` directories.
51. Primary Enterprise Server: Checks the status of the build and checks if BSFNs are complete. The Primary Enterprise Server checks the status of other servers if BSFNs are complete.
52. Primary Enterprise Server: When the dll and specs are finished, if the compress feature for server is enabled, it compresses the `.dll`, `.so`, `.sl`, and `.SRVPGM` files on the Primary Enterprise Servers. The Primary Enterprise Server sends a message to all the other servers to compress.
53. Other Enterprise Servers: Sends the compressed files to the Primary Enterprise Server under `<packagename>/<machine type>` directory.
54. Primary Enterprise Server: Transfers the compressed files and the `compress.inf` file back to the Deployment Server if initialization was successful under the `<packagename>/<machine type>` directory.
55. Deployment Server: When business function and business services are complete, the system starts the compression on the Deployment Server.
56. Primary Enterprise Server: Under the deployment location, creates the package name under the `bin64` and `specs` folder in these paths, respectively: `<pathcode>\bin64\<package name>` and `<pathcode>\spec \<package name>`.
57. Primary Enterprise Server: Copies the package `bin64` directory to the `<pathcode>\bin64\<package name>` directory.
58. Primary Enterprise Server: Under `<pathcode>\spec\<package name>`, creates the `spec.ini` file with the database datasource location for Central Objects **DV920_DataSource=Central Objects – DV920**
59. Primary Enterprise Server: Checks the `jde.ini` for entry

    ```
    [PACKAGE MANIFEST] CreateJAR=Y
    ```

    and/or checks the F96215->`PKGBULFUT3` = "`Y`" which is set when you select **Compress Application Component** in the Compress Options tab.

    If this entry is in `jde.ini` or F96215 with `'Y'`, then the system creates a `E1_APP_xxxx.jar` file under the package name directory on the Enterprise Server. This jar file contains the `bin64` directory and the `spec.ini` is populated with the package name. The jar file is used by the Server Manager to create a new Enterprise Server Instance.
60. Primary Enterprise Server: Transfers the server log (`svrpkgbuild.log`) from each server to the Deployment Server under `<packagename>\serverLogs` directory.
61. Primary Enterprise Server: Transfers the files located under the `text` directory on the each Enterprise Server to the Deployment Server under `<packagename>\serverlogs\<server name>\text`.

62. Primary Enterprise Server: Transfers the files located under the `CompileLogs` directory on the each Enterprise Server to the Deployment Server under `<packagename>\serverlogs\<server name>\CompileLog`.
63. Primary Enterprise Server: Transfers the files located under the `GenerateNER_logs` directory on the Primary Enterprise Server to the Deployment Server under `<packagename>\serverlogs\GenerateNER_logs`.
64. Primary Enterprise Server: If this is a server-only package, the process is complete. However, if client is selected, the system waits for the client to finish with the compression of the directories into CAB files.
65. Primary Enterprise Server: The system checks the status of the client build and the spec build to wait for them to complete. When complete, the process is done.

How the System Deploys from the Web

The P9631W web application calls B9600196 which calls into the enterprise server system code. The system then sends a message to the primary server to deploy the package to each server. It still follows the same steps described in the section *How the System Builds Packages from the Web (Release 9.2.5)* to deploy the package. These steps are all processed on the primary enterprise server.

Server Packages

This section discusses:

- A description of server packages.
- Primary server
- Jde.ini settings for server package builds.
- Spec.ini settings.
- Source code for Solaris servers.

A Description of Server Packages

A server package is a group of specification records, source files, header files, and compiled objects that are created on the enterprise servers. A server package is essentially the same as a client workstation package, with these exceptions:

- Foundation code is not deployed as part of a server package.
- All specs are built directly into the specified spec data source.

  The specs are copied from Central Objects to the database package tables.
- Some business functions (such as client only business functions) are not built on the server, and therefore are not included in a server package.

All application development takes place on workstations. Object-related files are stored in the F98780R repository table, and specs are stored in the central objects database on the database server. The application development life cycle is managed by Oracle's JD Edwards EnterpriseOne Object Management Workbench. This configuration enables you to partition business applications to an enterprise server. To ensure that modifications and enhancements that are developed on the workstation are reflected on the server, you must build a server package that contains those modifications and enhancements.

You use Oracle's JD Edwards EnterpriseOne Package Assembly (P9601/P9601W), Package Build Director (P9621/P9621W), and Package Deployment (P9631/P9631W) applications to assemble, define, build, and deploy server packages. After defining and building a server package, you can deploy it to an enterprise, logic, or application server by using Oracle's JD Edwards EnterpriseOne Deployment Director program (P9631/P9631W).

Primary Server

For an enterprise server build, there may be more than one enterprise server on which to build the server package. If there are multiple servers, one of these servers must be designated as the primary server.

The primary server controls the builds of the other enterprise servers. All source and include files are retrieved from the F98780R repository table on the primary enterprise server. The primary server then moves only the server-only business function source and include files to the other enterprise servers.

The primary server generates the NER, and builds the specs from Central Objects into the package database tables. Then it sends a message to the other enterprise servers to compile the business functions. It waits for all the servers to finish and then sends a message to compress-if compression was selected. The primary server is the only server to communicate with the client machine.

Build Net Changed Objects (Release 9.2.9)

Starting with Tools Release 9.2.9, if you select Build Net Changed Objects in Package Definition, the package will build all specs but only build business functions, tables, UBE with a `.h` file, and business view with a `.h` file that have changed since the last active package was built. The system copies all the files from the active package to the new package, retrieves from the repository only the objects with files which changed since the last active package, and builds the business function.

Also, the client package is initiated at the beginning of the server package build. The server and client package will run in parallel. This uses the Deployment Server Services regardless of whether it was started from a client/deployment machine or from the web.

Jde.ini Settings for Server Package Builds

If the server package includes business functions, the Build Settings within Server Manager apply to the package. This table describes various build settings:

Setting	Value	Purpose
Build Area	/usr/jdedwards/E920/packages	Indicates the location on the server where the package will be built.
Optimization Flags	+02 (default for HP-UX) -02 (default for AIX and Solaris)	Varies, depending on the platform. The system uses these compile flags to build business functions in Release mode. You should not change these flags.
DebugFlags	-g -D_DEBUG _DJDEDEBUG (default for HP-UX)	Varies, depending on the platform. The system uses these compile flags to build business

Setting	Value	Purpose
	-g -qfulpath -qdbextra -D_DEBUG -DJDEDEBUG (default for AIX) -g -D_DEBUG -DJDEDEBUG (default for Solaris)	functions in Debug mode. You should not change these flags.
InliningFlags	blank (default)	Indicates whether the *IBM i* uses inlining. Enter **Yes** to select inlining on the *IBM i*. Enter **No** to turn it off. This flag is blank or ignored for non- *IBM i* servers.
DefineFlags	-DKERNEL -DPRODUCTION_VERSION -DNATURAL_ALIGNMENT -D_HPUX-SOURCE (default for HP-UX) -DKERNEL -DPRODUCTION_VERSION -DNATURAL_ALIGNMENT (default for AIX) -DKERNEL -DPRODUCTION_VERSION -DNATURAL_ALIGNMENT -D_SUN-SOURCE (default for Solaris)	Indicates the Kernel Production Version of the source for HP-UX, AIX, and Solaris.
CompilerFlags	-Aa +w1 +z -c (default for HP-UX) -qalign=natural -qflag=l:l -c (default for AIX) -qspill=1024 -misalign -KPIC (default for Solaris)	Varies, depending on the platform. The spill flag sets the stack space when business functions are compiled. Typically, 1024 is adequate space to compile the delivered business functions.
OSReleaseLevel	+DAportable -q32 (for AIX)	Indicates the release level for which you are compiling the package. You should not change these flags.
LinkFlags	-b -z (default for HP-UX) -bl:/<your system directory>/bin32/functlist.imp -bM:SRE -bexpall -brtl -lc -bnentry -L. L/usr/<your system directory>/lib -ljdelib -ljdekrnl -ljdenet -bloadmap:loadmap (default for AIX) -G -L$(ORACLE_HOME)/lib (default for Solaris)	Varies, depending on the platform. The system uses these flags to link business functions. You should not change these flags.

Setting	Value	Purpose
	shared -z muldefs -melf_i386 -L/JDEdwards/e920/system/lib -ljdesaw (Linix-64 bit)	
LinkLibraries	blank (default)	Indicates the libraries to which business functions are linked. (Applies to Windows and *IBM i* servers only.)
SimultaneousBuilds	0 (unlimited) (default) any integer (number of simultaneous builds)	Indicates the number of DLLs that can be built at a time. Zero means that all will be built simultaneously.
Qname	queue name	Applies to *IBM i* only. Specify a queue name if you want the jobs for building dlls to go to a queue other than the default queue.
<compiler values>	Any compilers installed on the system will be retrieved from the registry and shown in a pull-down list. (Microsoft Windows platform)	Indicates the compiler level to use for builds. For more information on the supported versions of Visual Studio, see *the Visual Studio Requirements for Microsoft Windows-based Enterprise Servers section in the JD Edwards EnterpriseOne Tools Server Manager Guide* .
DeployWaitInMinutes	any integer (number of minutes)	Indicates the number of minutes to wait for an update or full package to deploy.

Spec.ini Settings

All JD Edwards EnterpriseOne servers and WIN32 clients require a spec.ini file to retrieve the package and spec data source information. This file is created by the Package Build process. It resides in the spec directory.

This table describes the settings within the SPEC LOCATIONS section of the spec.ini file:

Setting	Value
path code_Package	Indicates the name of the package.
path code_DataSource	Indicates the database data source that is used to retrieve the spec information.

Source Code for Solaris Servers

The Solaris compiler expects a newline character at the end of every source code file that it compiles. If the compiler does not find this newline character, it rejects the line and displays a warning message. In some cases, this line rejection and message might cause the package build to fail.

If you develop custom modifications on servers that use the Solaris operating system, you must ensure that this newline character is present in the compiled source code before you assemble, define, and build packages that contain the modifications. This step helps ensure that the package build process finishes successfully.

In some cases, the system automatically adds the newline character, and you do not need to add it manually. If you edit source code in the UNIX environment using an editor such as VI or emacs, these editors automatically add the newline character. Also, all of the source code files for business functions include the newline character.

However, editors that are included with PC workstations typically do not add the newline character. Therefore, if you edit source code on a PC workstation and then transfer the file to the server for compiling, verify that the newline character exists in the source code.

Note:

- *JD Edwards EnterpriseOne Tools Server Manager Guide*

Workstation Packages

This section discusses:

- Workstation installation.
- Building specifications and business functions.
- Defining the compiler level.
- Verifying UNICODE settings.
- Package INF files.

Workstation Installation

A typical full workstation installation takes more than 1.4 GB of disk space and can take 10 to 30 minutes to install, depending on network traffic. A workstation configuration contains the full suite of applications, including those that the user rarely or never uses, but all applications are available immediately.

Building Specifications and Business Functions

If you build a full client package that includes both business functions and specifications, add the following setting to the [INSTALL] section of the workstation jde.ini file on the computer that you use to build the packages:

```
WaitForBusbuild=Y/N
```

A *Y* means that business functions are built after all the specs are complete. The system builds the specifications and business functions sequentially instead of simultaneously.

An *N* means that specs and business functions are built simultaneously, which can speed up the build process.

Defining the Compiler Level

You must ensure that you have the highest currently supported compiler version as specified on Oracle Certifications for JD Edwards EnterpriseOne. You must specify the Visual Studio compiler level in the `jde.ini` file on the computer which you will run the package build. For example, if the supported version is Visual Studio 2022, you should specify the following in the `jde.ini` file:

```
[JDE_CG]
VisualStudioVersion=2022
```

If more than one supported compilers are installed on the computer that you use to build packages and the setting for `VisualStudioVersion` is not defined in the `jde.ini` file, the highest level compiler will be used to perform the build.

For more information on how to update this setting using Server Manager, as well as the supported versions of Visual Studio, refer to the Visual Studio Requirements for Microsoft Windows-based Enterprise Servers.

Improving Client Package Build Performance

If you find that client package builds take an inordinately long time when using Visual Studio 2013 or 2010, you can add and adjust the following setting in the [JDE_CG] section of the jde.ini on the build machine for the client package. The setting is passed to the Microsoft Visual C++ compiler as the /MP flag and specifies how many simultaneous compilations of business functions are allowed.

```
SimultaneousBuilds=X
```

In this example, the value of "X" is shown in the table:

Value of "X"	Description
0	This is the default value if the setting is missing. Visual Studio will attempt to start as many compilers as there are virtual threads on the build machine.
1	Only one compiler at a time will be run.
An integer less than or equal to the number of virtual threads on the build machine.	Visual Studio will attempt to start as many compilers as possible up to and including this number.

You can start with a value of zero (0), build a package, and, by looking at the build logs, determine how long the compilation takes. In some cases, you may not want to use all available virtual threads on the build machine. In that case, retry the build with various values until you get satisfactory build times.

Verifying UNICODE Settings

If you are upgrading from Xe and you have modified any interactive or batch applications that contain NERs and that are language-enabled, you must ensure that the following setting is in the [INSTALL] section of the primary enterprise server and workstation jde.ini on the computer that you use to build the packages:

```
Unicode Conversion Codepage=<code_page_value>
```

In this setting, code_page_value is a valid value for the code page of the language-enabled application that contains NERs. For example, for Korean language the value would be:

```
Unicode Conversion Codepage=ksc-5601
```

Note: If you are a language customer but have never added NERs to your applications, then you are not required to have this setting.

Package INF Files

The Package INF file is essentially the interface between the package build and Oracle's JD Edwards EnterpriseOne Workstation Installation program. The INF file defines the components that are included in the package, the source and destination paths for those components, and the attributes that are needed to install the package.

The INF file is created during the package build process and is stored in its own package_inf directory, based on the release master directory. JD Edwards EnterpriseOne Workstation Installation reads the INF file for the package that it is installing to determine which components are loaded to a workstation, as well as their locations.

Here is a typical INF file for package DV920FA, which is full package A for the DV920 path code. This INF file includes these sections:

- [SrcDirs]
- [DestDirs]
- [FileSets]
- [FileSetsDescription]
- [Components]
- [Typical]
- [Compact]
- [Attributes]
- [Oracle Databases]
- [Start]
- [Desktop]
- [Environment]
- [Fonts]
- [Feature]

- [Language]

[SrcDirs]

The JD Edwards EnterpriseOne Workstation Installation program uses these settings to determine the source path from which information is copied. JD Edwards EnterpriseOne Package Build compresses these directories. JD Edwards EnterpriseOne Workstation Installation copies the compressed directories to workstations.

Item	Purpose
SPathcode=\\MachineName \E920\PACKAGE\ DV920FA	Indicates the location of the package that the package builds for workstation installation. The default value for this path is the path code directory over which you built the package. You can change this setting if you want to use a different package.
SSYS=\\MachineName\E920\SYSTEM	Indicates the location of the system directory that the package builds for workstation installation. The default value for this path is the system directory that is associated with the path code over which you built the package. Normally, this directory is subordinate to the release share name (E920). This item appears only when included in the package.
SPathcodeDATA=\\MachineName\E920\ DV920\PACKAGE\DATA	Indicates the location of the Supported Local Database that the package builds for workstation installation. The default value for this path is the data directory that is subordinate to the path code over which you built the package. This item appears only when included in the package.

[DestDirs]

The JD Edwards EnterpriseOne Workstation Installation program uses these settings to determine the destination paths on the workstation. The process replaces %INSTALL with the user's computer configuration, which is set up in the User Display Preferences table (F00921) and the User Defined Codes Language Status table (F00051).

Item	Purpose
DPathcode=%INSTALL\path code	Indicates the destination directory for the package.
DSYS=%INSTALL\system	Indicates the destination directory for system files. This item appears only when included in the package.

[Filesets]

These settings list the various source and destination directories that are subordinate to the path code for the package. Y equals compressed, and N equals not compressed. The source and destination directory names are preceded by an *S* and *D*, respectively.

Item	Purpose
Pathcode1=Y, $Spathcode\bin32, $Dpathcode\bin32	Indicates the bin32 directory that is subordinate to the path code.
Pathcode1=Y,$Spathcode\bin64, $Dpathcode\bin64	Indicates the bin64 directory that is subordinate to the path code (Release 9.2.3)

Item	Purpose
Pathcode2=Y, $Spathcode\spec, $Dpathcode\spec	Indicates the spec directory that is subordinate to the path code.
Pathcode3=Y, $Spathcode\include, $Dpathcode\include	Indicates the include directory that is subordinate to the path code.
Pathcode3=Y, $Spathcode\include64, $Dpathcode\include64	Indicates the include64 directory that is subordinate to the path code.
Pathcode4=Y, $Spathcode\lib32, $Dpathcode\lib32	Indicates the lib32 directory that is subordinate to the path code.
Pathcode4=Y, $Spathcode\lib64, $Dpathcode\lib64	Indicates the lib64 directory that is subordinate to the path code.
Pathcode5=Y, $Spathcode\obj, $Dpathcode\obj	Indicates the obj directory that is subordinate to the path code.
Pathcode5=Y, $Spathcode\obj64, $Dpathcode\obj64	Indicates the obj64 directory that is subordinate to the path code.
Pathcode6=Y, $Spathcode\source, $Dpathcode\source	Indicates the source directory that is subordinate to the path code.
Pathcode6=Y, $Spathcode\source64, $Dpathcode\source	Indicates the source64 directory that is subordinate to the path code.
Pathcode7=Y, $Spathcode\work, $Dpathcode\work	Indicates the work directory that is subordinate to the path code.
Pathcode8=Y, $Spathcode\make, $Dpathcode\make	Indicates the make directory that is subordinate to the path code.
Pathcode9=Y, $Spathcode\res, $Dpathcode\res	Indicates the res directory that is subordinate to the path code.
Pathcode10=Y,$Spathcode\sbf.cab, $Dpathcode\java	Indicates the java directory that is subordinate to the path code.
SYS=Y,$SSYS\System.cab, $DSYS	Indicates the compressed system database that the package build generates.
PathcodeDATA=Y, $SpathcodeDATA \data.CAB, $DpathcodeDATA	Indicates the compressed data database that the package build generates.

[FileSetsDescription]

This section provides a text description for each fileset as shown in this example:

```
[FileSetsDescription]
DV9201=Business Function DLL Files
DV9202=Specification Files
DV9203=Include Files
DV9204=Library Files
DV9205=Object Files
DV9206=Source Files
DV9207=Work Files
DV9208=Make Files
DV9209=Resource Files
DV92010=SBF Source Files
SYS=Foundation Files
```

Or for 64-bit packages (Release 9.2.3):

```
[FileSetsDescription]
DV9201=Business Function 64-bit DLL Files
DV9202=Specification Files
DV9203=Include 64-bit Files
DV9204=Library 64-bit Files
DV9205=Object 64-bit Files
DV9206=Source 64-bit Files
DV9207=Work Files
DV9208=Resource Files
DV92010=Include Files
DV92011=Source Files
SYS=Foundation Files
```

[Components]

These settings indicate the location of the foundation, production, and development objects, as well as the database files.

Item	Purpose
Production Objects=APPL_PKG1, APPL_PKG2, APPL_PKG3	Indicates the location of production objects.
Development Objects=APPL_PKG4, APPL_PKG5, APPL_PKG6, APPL_PKG7, APPL_PKG8, APPL_PKG9	Indicates the location of development objects.
Foundation=SYS	Indicates the foundation location.

[Typical]

This section describes the setting for a typical development user.

Item	Purpose
Name=Development	Indicates that the package is for a development user.
Description=Install the development objects	Indicates the package description.
Components=ProdObj, DevObj, Foundation	Indicates that the package should contain both production and development objects, as well as the foundation.

[Compact]

This section describes settings for a typical production user.

Item	Purpose
Name=Production	Indicates that the package is for a production user.
Description=Install the production objects only	Indicates the package description.
Components=Production Objects, Foundation	Indicates that the package should contain only production objects, as well as the database

[Attributes]

This section contains information about the current release, global tables, specification and help files, and the jde.ini file.

Item	Purpose
PackageName=DV920FA	Indicates the name of the package.
PathCode=DV920	Indicates the path code for which the package is being built.
Built=Build Completed Successfully	Indicates the package status. A status of 50 or 70 means that the package is ready for installation.
PackageType=Full	Indicates the package type: full or update.
SPEC_FORMAT=XML	Indicates the format for the specifications.
Release=E920	Indicates the current release, which determines the setup.inf file to use in building the jde.ini for the workstation. The release is also used to determine paths for system and helps.
BaseRelease=B9	Indicates the current base release.

Item	Purpose
SystemBuildType=RELEASE	Indicates the type of build: DEBUG or RELEASE. This option is retrieved from owver.dll.
ServicePack=01-15-2018_08_43 - Release64	Indicates the date that the Tools Release was built, the mode (debug or release), and the bitness. (Release 9.2.3)
ToolsBitness=32 orToolsBitness=64	Indicates the bitness of the package (Release 9.2.3)
MFCVersion=6	Indicates the version of the MFC compiler.
SpecFilesAvailable=Y	Indicates that specification files are available to merge or copy. This option is always set to Y for full and update packages.
DelGlbTbl=Y	Indicates whether to delete global tables when installing. This option is set to Y for full packages. For update packages, this option is set to Y only if the objects include a table object.
ReplaceIni=Y	Indicates whether to delete the existing jde.ini file when installing, and then create a new one. This option is set to Y for full packages. For update packages, the user must specify during package build definition whether to replace the jde.ini file.
AppBuildDate=Mon Jul 20 11:22:22 2020	Indicates the date and time when the package was built. Full packages always have this date. This option is blank when no objects are included in the package.
FoundationBuildDate=Wed Jun 03 15:08:34 2020	Indicates the date and time when the foundation was built.The date is retrieved from owver.dll. Full packages always have this date. This option is blank when no foundation location is specified in the package.
DataBuildDate=Wed Jun 03 15:08:34 2020	Indicates the date and time when the database file was built. Full packages always have this date. This option is blank when no database location is specified in the package.
DeploymentServerName=DENMLSAN222	Indicates the name of the deployment server.
Location=DENVER	Indicates the location of the deployment server.
DeploymentStatus=Approved	Indicates the package deployment status.
PackageDescription=Development full package A	Indicates the package description.
Icon Description=JDEdwards	Describes the desktop icon.
Default Environment=DV920	Indicates the default environment.
AllPathCodes=Y	Indicates that a package for *ALL path codes exists when set to Y.

Item	Purpose
Spec=	Indicates the format of the specifications with these values: • XML_RDBMS Use this setting when XML specs are in an RDBMS (full package). • XML_TAM Use this setting when XML specs are in TAM format (update package).
Language	Lists the languages in the package using language codes and separated by commas.
BuildLocalDatabase (Tools Release 9.2.5.0)	Indicates a local database. N means the local database was not built. Y means there is a local database. If this is not present then Y is assumed.
SpecSrcPath (Tools Release 9.2.5.0)	Path to the package spec folder with the .par files. This is used to delete objects specs for apps from the spec\runtime folder.

[Oracle Databases]

This section contains information about the Oracle databases.

Item	Purpose
JDELocal_DV920=ORACLE	Indicates the local Oracle database.
SPEC_DV920FA=ORACLE	Indicates the Oracle database.

Each of the databases listed, has its own section in the INF file as shown in this example:

```
[JDELocal_DV920]
SourceTableSpace=JDELocal
Server=127.0.0.1
UserID=SYSTEM
DataFileDestDir=$DDV920DATA\JDELocal_DV920.dbf
DumpFileDestDir=$DDV920DATA\JDELocal_DV920.dmp

[SPEC_DV920FA]
SourceTableSpace=MASTER
Server=127.0.0.1
DataFileDestDir=$DDV920\Spec\SPEC_DV920FA.dbf
DumpFileDestDir=$DDV920\Spec\SPEC_DV920FA.dmp
```

[START]

This section contains startup information:

```
ProgramGroupName=JDEdwards
Item1=System\Bin32\activConsole.exe, JDEdwards Solution Explorer, appl_pgf\res\One#
World.ico
```

Or for 64-bit (Release 9.2.3):

```
ProgramGroupName=JDEdwards
Item1=System\Bin64\activConsole.exe, JDEdwards Solution Explorer, appl_pgf\res\One#
World.ico
```

[Desktop]

This section contains desktop information:

```
Item1=System\Bin32\activConsole.exe, JDEdwards Solution Explorer, appl_pgf, res##
\OneWorld.ico
```

Or for 64-bit (Release 9.2.3):

```
Item1=System\Bin64\activConsole.exe, JDEdwards Solution Explorer, appl_pgf, res##
\OneWorld.ico
```

[Environment]

This section contains environment information:

```
PathDV920=%INSTALL\DV920\bin32;
PathSys=%INSTALL\system\bin32;
```

Or for 64-bit (Release 9.2.3):

```
PathDV920=%INSTALL\DV920\bin64;
PathSys=%INSTALL\system\bin64;
```

[Fonts]

This section contains font information:

```
[Fonts]
Arial=Font\arial.ttf
```

[Feature]

This section describes information for any features that are included in the package. A feature is a set of files or configuration options that is included in a package for deployment to a workstation or server. This example shows some features that might be included:

```
[Feature]
WEBDEVELF=\\DENMLSAN246\E920\Package_inf\Feature_inf\WEBDEVELF_1.INF
VS2010RT=\\DENMLSAN246\E920\Package_inf\Feature_inf\VS2010RT_1.INF
```

[Language]

This section describes information for all of the languages that were included in the package. This example includes simplified Chinese and German:

```
[Language]
LANGUAGE=CS,G
```

If there are no languages, then the value is blank:

```
[Language]
LANGUAGE=
```

Files Created by the Build Process

This section discusses:

- Workstation package build
- Server package build
- UNIX server build
- Windows server build
- *IBM i* server build

Workstation Package Build

Business function dynamic linked libraries (DLLs) on workstations are grouped by related business functions. This grouping limits the size and number of procedures that are contained in each DLL. Grouping prevents memory allocation errors and avoids the platform limitations that can occur when you export too many procedures from the same DLL.

The production environment PD920/bin32 directory contains the DLLs that are created on the workstation. All of the business function source files are in the PD920/source directory.

Files Created by a Business Function Build

When you build a single business function through Oracle's JD Edwards EnterpriseOne Object Librarian, the Business Function Builder program uses the make (*.mak) file that is generated at runtime, and creates or copies these files and builds the business functions into their respective DLLs:

- Source file (*.c)
- Header file (*.h)
- Object file (*.obj)

You must use the jdecallobject API to call a business function from a business function.

These files are created for NER business functions:

- OBJNAME.c
- OBJNAME.h
- OBJNAME.obj

These files are created for table event rules:

- OBJNAME.c
- OBJNAME.hxx
- OBJNAME.obj

Server Package Build

Server package builds are used to move path code objects stored in the F98780R repository table to enterprise server platforms. Server package builds are initiated when you create either full or update packages during package assembly. After you have assembled the package, you must select the server option during package definition, and select the relevant servers from the list of available servers in the screen that follows. When package definition is complete and the package has been activated, highlight the package and select Submit Build from the Row menu to start the server package build. When the server package build has finished successfully, you can deploy the server package.

To assemble a server package, use the foundation, database, and object information in package assembly to generate build information, specification files, and business function source for .c, .h, and .hxx files. After the server package build has generated these objects and placed them in the staging area, the system transfers the objects to each of the servers that is specified in the package definition. The system then directs the servers to compile the business function source code and generate the corresponding business function DLLs.

UNIX Server or Windows Server Build

This topic describes the files that the system creates when it builds business functions on a UNIX or Windows server.

Files Created by a Business Function Build

When you are building business functions, these groups of source files are actually compiled:

- NER business functions.
- Table event rules.
- C business function event rules.

When you are building business functions, these file types are supplied to the build process:

- Source files (.c)
- Header files (.h, .hxx)

When building business functions, the build process creates these file types:

- Object files (.o)
- Make files (.mak)
- Shared libraries (.sl, .so) for UNIX or .dll for Windows

Shared libraries for business functions, which are equivalent to a DLL for a Windows workstation, are consolidated. Therefore, one shared library is created for each parent DLL in the Object Librarian - Status Detail table (F9861). If you are creating custom business functions, use a custom parent DLL instead of one of the parent DLLs that JD Edwards EnterpriseOne software provides.

Where Business Functions Are Stored

On a UNIX or Windows platform, related business functions are grouped into shared libraries. This grouping limits the size and number of procedures that are contained in each shared library. Grouping prevents memory allocation errors and avoids platform-specific limitations in the number of procedures that you can export per shared library.

The exact location of the package is determined by the Build Settings in the jde.ini which can be accessed through Server Manager.

Subordinate to the package directory (PD920FA) is a source directory. This source directory contains subdirectories for each shared library that is created on the enterprise server.

The checkin location holds all the include, source, and res files that were retrieved from the F98780R during the build. The process then copies only the server-only or server/client business functions to the PD920FA source and include directories to prepare it to build the business functions.

The directory structure looks like this example where the top directory represents the package name:

```
PD920FA\source\CAEC

PD920FA\source\CALLBSFN

PD920FA\source\CCORE

PD920FA\source\CDESIGN

PD920FA\source\CDIST

PD920FA\source\CFIN

PD920FA\source\CHRM

PD920FA\source\CMFG

PD920FA\source\JDBTRIG

checkin\include and/or include64

checkin\source and/or source64

checkin\res
```

Each subdirectory contains the business function source files that belong to the shared library. All shared libraries are installed in the PD920/bin32(bin64) directory.

The naming convention for UNIX for the shared libraries is as follows: lib, followed by the name of the shared library subdirectory, followed by .sl (for HP-UX) or .so (for LINUX and AIX). An example is libccore.so.

For Windows, all DLLs are installed in the PD920\bin32(bin64) directory. They have the same name as the DLL subdirectories, except that they have the .dll suffix.

Specification Files

JD Edwards EnterpriseOne specifications are stored in an RDBMS. The database data source for this database is specified in the spec.ini file or is selected by the package build administrator during the server package build definition process. Package Build copies the specs directly from the build machine to the specified spec database. However, local cache (GLBLTBL, DDDICT, and DDTEXT) specification files are still created in TAM format. The contents in these files are destroyed when a new package is deployed.

Server Build

This topic describes the files that the system creates when it builds business functions on an *IBM i* server.

Files Created by a Business Function Build

Starting with Tools Release 9.2.6, when building business functions, the server package build creates these file types in unique library whose name is created with the first 2 letters and the last 2 letters of the package name, and the job number. For example, for package DV920FA, the unique library would be DVFA<job number>. This unique library is in the QSYS file system and contains:

- *MODULES - These are object files.
- *USRSPC - User spaces hold information about which .c files each business function DLL contains.
- *SRVPGM - Server programs are the DLLs on the *IBM i* .
- *FILE - Contains only the logs about compiled business functions.

Starting with Tools Release 9.2.6.0, the active or deployed location is used as the package name. During the build, after building the business functions the package copies the MODULES, SRVPGM, and FILE from the unique library to the package name library in the QSYS file system. This action sets up the package for deployment.

(Tools Release 9.2.7.3) When the package is deployed, if the *IBM i* instance has an addendum associated with it, the *SVRPGM, *MODULE, and *USRSPC in the package name library are copied to a new library called `<packagename><addendum>`. The character limit of the package name is limited to 10 minus <addendum length>= package name length. This ensures that the name of the package along with the addendum will be 10 or less characters to create the new library name.

Where Business Function Source Members Are Stored

IBM i business function source and headers are now transferred to the Integrated File System (IFS). Server package build transfers objects to these subdirectories under the server package directory in the IFS for the *IBM i* .

The exact location of the package is determined by the Build Settings in the jde.ini which can be accessed through Server Manager.

Subordinate to the package directory (PD920FA) is a source directory. This source directory contains subdirectories for each DLL that is created on the enterprise server.

The directory structure looks like this example where the top directory is the package name:

```
PD920FA

include

source

CAEC

CALLBSFN

CCORE

CDESIGN

CDIST

CFIN

CHRM

CMFG
```

`JDBTRIG`

`spec`

`text`

`checkin\include and/or include64`

`checkin\source and/or source64`

`checkin\res`

> **Note:** After an upgrade, existing *IBM i* server path codes must be rebuilt with the server package build to avoid problems building server package updates and manually re-linking business functions using the LINKBSFN program. Starting with Tools Release 9.2.6, during the component swap in Server Manager, you have an option to link the business functions automatically.

Specification Files

JD Edwards EnterpriseOne specifications are stored in an RDBMS. The database data source for this database is specified in the spec.ini file or is selected by the package build administrator during the server package build definition process. Package Build copies the specs directly from the build machine to the specified spec database. However, local cache (GLBLTBL, DDDICT, and DDTEXT) specification files are still created in TAM format. The contents in these files are destroyed when a new package is deployed.

Descriptions of Directory Files

This table describes the files that are found in the directories:

Directory	Description
`PD920\include`	This is the location where .h and .hxx source files are located. These objects are taken from the server and built on.
`Checkin\include and/or include64`	This is the location where all the .h include files retrieved from the F98780R repository are stored.
`Checkin\source and/or source64`	This is the location where all the .c source files retrieved from the F98780R repository is stored.
`Checkin\res`	This is the location where all the bitmaps used by the applications are stored.
`PD920\source`	This directory contains subdirectories that include the business function DLL names. Each subdirectory contains .c source for the business functions that are compiled and linked into the DLL.
`PD920\spec`	This folder is no longer used but is created for backwards compatibility.
`PD920\text`	This directory contains build text, status files and log files (.txt, .sts, .log) for business function DLLs and specification files. The text files contain information that is needed for the server package build. The text files also contain build directives for creating business function DLLs. The status files for specification files indicate whether a server package build was successful in converting pack files into

Directory	Description
	spec files. The status files for business function DLLs indicate which .c source files were successfully compiled and linked. The log files created exclusively for business function DLLs contain the compiler commands used to build and link business functions. For the Microsoft Windows platform, the beginning of this file identifies the compiler used to perform the build.

Features

This section discusses:

- Defining features
- Feature INF files

Defining Features

In addition to objects, you can also add a *feature* to a package. A feature is a set of files or configuration options that must be copied to a workstation or server to support an application or another function. Like objects, features are included in a package and deployed to the workstations and servers that require the feature components.

For example, you might need to add these items to a package: ActiveX controls, a Supported Local Database for the Sales Force Automation feature, ODBC data sources for use with Open Data Access, or Microsoft Windows registry settings.

You define a feature by using the JD Edwards EnterpriseOne Package Assembly program (P9601). You can then add the feature to a package by using the JD Edwards EnterpriseOne Package Assembly program (P9601) and Package Build Director program (P9621).

Feature INF Files

When a package contains features, a section called [Features] in the Package INF file includes both the feature name and a pointer to the Feature INF file that is created for each feature in the package. These Feature INF files provide specifications that tell the installation program the actions to perform during the installation.

The Feature INF file can include these sections:

- [Header]
- [Registry]
- [INI]
- [FileSets]
- [Shortcut]
- [ThirdPartyApps]
- [ODBCDataSources]

This is a typical Feature INF file for which the sections contain specifications for each feature component.

[Header]

The header section contains general information about the feature and specifies the installation options for the feature.

Item	Purpose
Feature=	Name of the feature.
FeatureType=	Type of feature.
Description=	Text description of the feature.
Required=	A setting that indicates whether installation of the feature is required.
InitialChoice=	A setting that specifies the default selections for features that the user can install.

The Required and InitialChoice entries are set using the three Feature Installation option settings (Required, Selected, Deselected) on the Feature Information form. When you select one of these three options, the system writes these values into the Required and InitialChoice entries in the feature INF file.

Feature Installation Option	Required	InitialChoice
Required	Y	Both
Selected	N	Both
Deselected	N	Custom

[Registry]

This section contains information about how the feature affects the Windows registry.

The settings for this section are displayed in this order:

Registry_no.=Root[value],Key,[prefix]Name,[prefix]Value

The following table contains a description of each variable:

Item	Purpose
Root	Describes the root in the registry with these values: • 0 means root • 1 means current user • 2 means local machine location

Item	Purpose
	• 3 means users
Key	Indicates the key for the registry value.
Name	The registry value name. Name prefixes are: • + means that the name is created (if it does not already exist) when the feature is installed • – means that the name is deleted with all subkeys when the feature is uninstalled • * means that the name is created (if it does not already exist) when the feature is installed, and it is removed with all subkeys when the feature is uninstalled
Value	The name of the registry value. Value prefixes are: • #x means that the value is stored as a hexadecimal value • #% means that the value is stored as an expandable string • # means that the value is stored as an integer • #$ means that the value is stored as a string

[INI]

This section contains information about how the feature affects the jde.ini file.

The settings for this section are displayed in this order:

Ini_no.=FileName,Directory,Section,Key,Value,Action

The following table contains a description of each variable:

Item	Purpose
FileName	The name of the destination INI file.
Directory	The location of the destination INI file.
Section	The name of the section in the destination file.
Key	The name of the key within the section of the destination file.
Value	The value to be written to the key of the destination file.
Action	The action to take regarding the INI entry: • 0 means create the INI entry. • 1 means create the INI entry only if it does not already exist. • 3 means create the INI entry or append to the existing entry.

[FileSets]

This section contains information about additional files that must be installed for the feature to function correctly.

The settings for this section are displayed in this order:

Fileset_no.=Compression,SourceDirectory,FileName,TargetDirectory

The following table contains a description of each variable:

Item	Purpose
Compression	An option that indicates whether the fileset is compressed.
Source Directory	The source location of the fileset.
FileName	The name of the CAB file for the fileset.
Target Directory	The target location into which the fileset will be placed.

[Shortcut]

This section contains information about shortcuts that appear on the Windows desktop as part of the feature installation.

The settings for this section are displayed in this order:

Shortcut_no.=Directory,Name,Target,Arguments,Description,HotKey,Icon,IconIndex,ShowCmd,WKDir

The following table contains a description of each variable:

Item	Purpose
Directory	The directory where the shortcut is created.
Name	The name of the link file for the shortcut.
Target	The name of the executable file for the shortcut.
Arguments	Any command line arguments for the shortcut.
Description	A description of the shortcut.
HotKey	A hot key that launches the shortcut.
Icon	The shortcut icon and location.

Item	Purpose
IconIndex	An index of the icon if the icon is inside an image list.
ShowCmd	A command for the application window, with these value options: • 0 means show the window normal-sized. • 3 means show the window maximized. • 7 means show the window minimized; not active.
WkDir	The working directory for the shortcut.

[ThirdPartyApps]

This section contains information about third-party products that are installed with the feature.

The settings for this section are displayed in this order:

ThirdPartyApp_no.=Source Directory,Description,Synchronous/Asynchronous,Execute Before/After,FileName

The following table contains a description of each variable:

Item	Purpose
Source Directory	Source location of the executable for running the third-party application.
Description	Description of the third-party application.
Synchronous/Asynchronous	An option that indicates whether the third-party application can be installed in parallel (synchronous) or must be installed serially (asynchronous).
Execute Before/After	An option that indicates whether the third-party application installation is run before or after JD Edwards EnterpriseOne is installed.
FileName	The name of the file that launches the third-party application.

[ODBCDataSources]

This section contains information about ODBC data sources that are installed with the feature.

ODBC data sources have two sections in the feature.inf. One section contains header information and the other contains the detail information. The feature.inf contains one header section listing all data source components that are included in the feature. For each data source that is listed in the header, a corresponding detail section exists. Only the header section is described in this table. For information about the detail section, see the documentation for the selected ODBC Driver.

The settings for this section are displayed in this order: DataSourceName=DataSourceDriver

Item	Purpose
DataSource Name	The name of the ODBC data source.
DataSource Driver	The driver that is used for the data source.

6 Building Packages

Understanding the Package Build Process

After you assemble a package, you must define the package build before you can build and deploy it to the workstations and servers. The build process reads the central objects data source for the path code that you defined in the package. This information is then converted from a relational database format to replicated objects, which are put in the package itself.

This section discusses:

- Directory structure for packages.
- Package build tasks.
- JD Edwards EnterpriseOne Package Build Definition Director.
- Business function builds during package build.
- Package compression.
- Verification of a package build.

Directory Structure for Packages

When a package is built, a directory structure with the name of the package is created within the appropriate path code directory. This directory contains the package information.

Example: JD Edwards EnterpriseOne Deployment Server Directory Structure

The E920 (release name) directory structure looks similar to this example.

This is an example of the directory structure for PD920 (path code name)\Package\<package name> on the Deployment Server:

`bin64`

`include`

`java`

`lib32`

`obj`

`pkgspec` (Prior to Tools Release 9.2.5.0)

`res`

`serverlogs`

`source`

`spec`

`work`

`<operating system name of enterprise server (for example, INTEL5.2)`

This is an example of a directory structure that has a 64-bit enabled path code:

`lib32 OR bin64`

`include`

`include64`

`java`

`lib32 OR lib64`

`obj OR obj64`

`pkgspec` (Prior to Tools Release 9.2.5.0)

`res`

`serverlogs`

`source`

`source64`

`spec`

`work`

When you build a client-only package, the directories under the package name are populated. Files for the source/ source64 and include/include64 directories are retrieved from the repository, F98780R table. For server/client packages, the include and source are copied from the primary build server checkin/include for 32-bit, checkin/ include64 for 64-bit and checkin/source for 32-bit, checkin/ source 64 for 64-bit folders. Information for all other directories comes from central objects. The bin32/bin64, lib32/lib64, and obj/obj64 directories are populated with the output of the business function build process.

Example: JD Edwards EnterpriseOne Enterprise Server Directory Structure

The E920 (release name) directory structure looks similar to this example.

> **Note:** Any references to 64-bit apply to Release 9.2.3 or higher only.

This is an example of the directory structure for PD920 (path code name)\Package\<package name> on the Enterprise Server:

`bin32 OR bin64`

`CompileLogs`

`compress`

`GeneratedNER_logs`

`include`

`checkin\include`

`checkin\include64`

```
lib32 OR lib64

obj OR obj64

res

checkin\res

source

checkin\source

checkin\source64

spec

text

work

<operating system name of enterprise server (for example, INTEL5.2)
```

Package Build Tasks

The process that you perform to build a package might take several hours. For this reason, it is recommended that you initiate the actual package build at the end of the working day, if possible. Complete these tasks when you build a package:

- Transfer objects.

  Ensure that all of the objects that you want to include in the build have been transferred to the appropriate path code.
- Ensure that the database for the package has the most current replicated data.
- Build a package.

  Build a package using the path code to which objects were transferred.
- Deploy the software to these machines:
  - Workstations and servers
  - Tiered deployment locations

CAUTION: If in the jde.ini, debug logging is set to perform logging, the package build process will take even longer to complete. Logging should be turned off during package builds to improve performance.

Package Build Definition Director

Like the Package Assembly Director, the Package Build Definition Director simplifies and expedites the build definition process by displaying a series of forms that guide you through the process. As with the Package Assembly Director, you can either click Next to continue to the next form or Previous to go back to the previous form. You can cancel the build definition process by clicking Cancel.

Like Oracle's JD Edwards EnterpriseOne Package Assembly application, Oracle's JD Edwards EnterpriseOne Package Build application defaults to Express mode. You can use a processing option to switch the JD Edwards EnterpriseOne Package Build application to Director mode.

This table summarizes the function of each form in the Package Build Definition Director:

Form	Description
Package Build Definition Director	Use this form to review introductory information about the Package Build Definition Director.
Package Selection	Use this form to select the defined package that you want to build. The status of the package must be **Assembly-Definition Complete.**
Package Build Location	Use this form to specify whether you want to build the package for the client workstation, one or more servers, or both clients and servers. For server packages, also specify the shared specifications data source.
Server Selection	Use this form to specify the server location. (The server location is required when you build a package for a server.) If you only select one server, this server becomes the primary build server. If you select more than one server, you must also select the primary build server from the list of servers.
Build Specification Options	Use this form to specify whether you want to include all specification tables or only selected tables in the package. The option to build individual specifications is useful if a build fails and the package error log indicates that an individual specification file needs to be rebuilt.
Individual Specification Selection	Use this form to include only selected specifications in the package.
Business Function Options	Use this form to build business functions. You can also specify the build mode, the severity level at which to interrupt the build process, whether to build business function documentation, and whether to clear the output destination before building.
Compression Options	Use this form to specify package compression for client or server packages and to specify whether to compress directories (all or individual), data, and foundation.
Individual Directory Selection	Use this form to select the individual directories that you want to compress.
Build Features	Use this form to enter file set and compression information for features that you added to the package. A feature is a set of files or configuration options, such as registry settings, that must be copied to a workstation or server to support an application or other function.
Package Build Revisions	Use this form to review or change any of the options that you have specified for the package.

You can access the Package Build Definition Director from either the Package Assembly menu selection or the Package Build menu selection. The advantage of accessing the Director from the Package Assembly menu selection is that the system automatically enters the package name and other information. If you access the Director from the Work With Package Build Definition form, you must manually specify the name of the package that you want to build.

Before you launch the Package Build Definition Director, you can use the Work with Package Build Definition form to review information about any previously designed packages. For example, you can review the properties, build options, business function options, and compression options for the package. As on any other parent/child form, you can click the plus (+) symbol to view more information about the package or click the minus (-) symbol to view less information about the package.

Viewing Package Build History and Resubmitting Builds

After you submit the package build, you can track the build status using Oracle's JD Edwards EnterpriseOne Package Build History program (P9622). This application also enables you to view logs that are associated with the build process to determine if any errors occurred during the build process.

If the build did not complete successfully, you can resubmit the package and resume building from the point where the build stopped. Alternatively, you can reset the status of the specifications and objects and then build the package again.

Note:

- *Viewing the Package Build History*.

Business Function Builds During Package Build

When you build business functions as part of the package build, the system performs the same process as if you had manually run Oracle's JD Edwards EnterpriseOne BusBuild program (selected Build from the Global Build menu) after you built the package.

Note: Any references to 64-bit apply to Tools Release 9.2.3 or higher.

The system retrieves source and header information from the package (from the source/source64 and include/include64 directories), compiles it, and stores it in the bin32, obj, and lib32 directories or bin64, obj64, and lib64 if building with a 64-bit package. The system builds business functions in the package, not on the workstation. If you select the Compress Package option, the system compresses the business functions after it builds them.

These guidelines apply to the path code, foundation, and destination for the business function build:

- When building business functions, use the path code that you defined in the package.
- The foundation is either the same as the foundation that is included in the package or, for an update package, it is the foundation for the parent package.
- Build output is directed to the bin32, obj, and lib32 directories of the package itself; or if building a 64-bit package, then the bin64, obj64, and lib64 directories of the package.
- When building a full package, or when building an update package that includes a business function, always build business functions; otherwise, the consolidated DLLs included in the package will not be current.

  For update packages, the system builds each business function individually. After it builds an individual business function, the system performs a global link for that object and all other objects that are in the same consolidated DLL. The global link affects all objects in the check-in location for the path code of that package.

Package Compression

You can compress a server package or client package.

Compressing Server Packages

If you plan to deploy a package to an enterprise server, which you did not build the package on, you must build the package on the same type of server, with compression selected. The directories are compressed into file types that are compatible with the type of enterprise server to which you are going to deploy. For NT servers, the file extension is .cab, for UNIX the file extension is .z, and for *IBM i* , the file has no extension.

When the server builds a compressed package, it stores the compressed files on the primary build server in subdirectories, such as \bin32 or bin64, then they are copied to the package path on the deployment server, \package \package name\server type\, where package name is the name of the package and server type is the type of server for which the package is compressed.

The compression process creates a new file called compressed.inf in the server type directory. This file includes the information that the system needs to deploy the compressed files. This table shows the type of compressed file that the process creates for each type of server:

NT	**UNIX**	***IBM i***
.cab	.z	SAVF

When you deploy the package to another enterprise server, the system reads the compress.inf file and uses this information to copy the compressed files from the package directory on the primary build server to the enterprise server.

Specs are no longer compressed. When they are deployed, the spec.ini is set to the package name for that path code.

Compressing Server Update Packages

When you select the option to compress the server package, the program creates a compressed file in the package name\server type\bin32 or bin64 directory. Specifications are not compressed. They are merged directly from the update package spec repository tables into the parent package spec repository tables.

Compressing Client Packages

When you compress directories, the application objects that are included in the package are automatically compressed. The system also creates an entry in the package INF file that indicates whether the foundation, data, and application objects are compressed.

Compressing Application Component (Release 9.2.7)

When you select the option to Compress the Application Component, the system compresses the bin32 or bin64 directory of the package (IBMi: SVPGM, MODULES, USRSPC, FILE) and the spec directory into a E1_APP_xxxx.jar file. This jar file can be uploaded and distributed in the Server Manager to the same OS Enterprise server.

Once the jar file is uploaded, you can go into the Enterprise Server Managed Home Instance and select Create Instance. This will create a new Enterprise Server Instance using the E1_APP_xxxx.jar file and a System Tools Release.

Note: *Click here to view a recording of this feature.*

Verification of a Package Build

After you assemble a package or define a package build, you can verify whether the package can be built successfully. You can use this verification to test the package before you submit the build, or troubleshoot problems with the build process if the package build fails.

During the verification process, the program verifies that:

- Disk space is adequate.
- Central objects and package build tables are accessible.
- User has permissions to create directories on the deployment server and enterprise server.
- Required service pack is installed.
- Machine tables are set up.
- Required compiler version is installed.
- Server port is accessible.
- Debug levels of the jde.ini files are adequate for the client and enterprise server.

Note: You cannot verify the specification database data source. Only enterprise servers and clients can be verified.

Building a Package

This section lists prerequisites and discusses how to:

- Set processing options for the Package Build Definition Director.
- Define a package build.
- Review package build selections.
- Build a package.

Prerequisites

Before you complete the tasks in this section:

- Verify that the User ID that is used to perform the package build has drop table and create table rights.

  If you are using a Oracle, SQL, or *DB2 for Linux, UNIX, and Windows database* , and you are running with security server turned on, you must add a security override so that the package build process can create the metadata repository table in central objects.

  See *Adding a Security Override for Package Build*.
- For customers adopting Microsoft Visual C++:

All JD Edwards EnterpriseOne Windows machines receiving application foundation packages built with Microsoft Visual C++ require the corresponding runtime libraries to be installed.

- Assemble the package and verify that the status of the assembled package is **Assembly-Definition Complete.**
- Verify that Oracle's JD Edwards EnterpriseOne Object Configuration Manager (OCM) mappings are correctly set for the JD Edwards EnterpriseOne Package Build (R9621S) and Server Package Build (R9622C) programs, which the system generates as part of the package build process.

  For example, if you want the programs to run locally, ensure that the OCM mappings point to Local for the environment in which the package build is running.
- Verify that logging is turned off during the package build.

  When the jde.ini file is set for logging in the c:\windows folder and a package build is submitted, the build process is slowed down.

Note:

- *"Understanding and Working with Object Configuration Manager" in the JD Edwards EnterpriseOne Tools System Administration Guide*

Forms Used to Build a Package

Form Name	FormID	Navigation	Usage
Package Selection	W9621C	Package and Deployment Tools (GH9083), Package Build. Click Add, and then click Next.	Select a package to define a build.
Package Build Revisions	W9621B	Package and Deployment Tools menu (GH9083), Package Build. Select a package, and deactivate it by selecting Active/Inactive from the Row menu. Select Build Revisions from the Row menu.	Review and change package build options.
Work With Package Build Definition	W9621L	Package and Deployment Tools (GH9083), Package Build	Add a package build definition. Build a defined package.

Setting Processing Options for the Package Build Definition Director (P9621)

Processing options enable you to specify the default processing for programs and reports.

For programs, you can specify options such as the default values for specific transactions, whether fields appear on a form, and the version of the program that you want to run.

For reports, processing options enable you to specify the information that appears on reports. For example, you set a processing option to include the fiscal year or the number of aging days on a report.

Do not modify JD Edwards EnterpriseOne demo versions, which are identified by ZJDE or XJDE prefixes. Copy these versions or create new versions to change any values, including the version number, version title, prompting options, security, and processing options.

Processing Tab

Although processing options are set up during JD Edwards EnterpriseOne implementation, you can change processing options each time you run a program.

1. Changes
Enter a value to determine how changes will occur.

<Blank> means that changes will only be allowed at the package level and will apply to all servers selected.

Enter **1** to enable changes to the build definitions by individual server.

2. Mastering
Mark this processing option with a **1** if this process is for Mastering purposes. If the process is for all users, mark this processing option with **<Blank>.**

3. Build Verification
Mark this processing option with a **1** if the Build Verification UBE is to be run prior to building all packages. If the build verification fails, the package build UBE will not be run. Leave this processing option **<Blank>** if you do not want to run Build Verification.

4. Director or Express Mode
Use this processing option to switch between Director and Express modes.

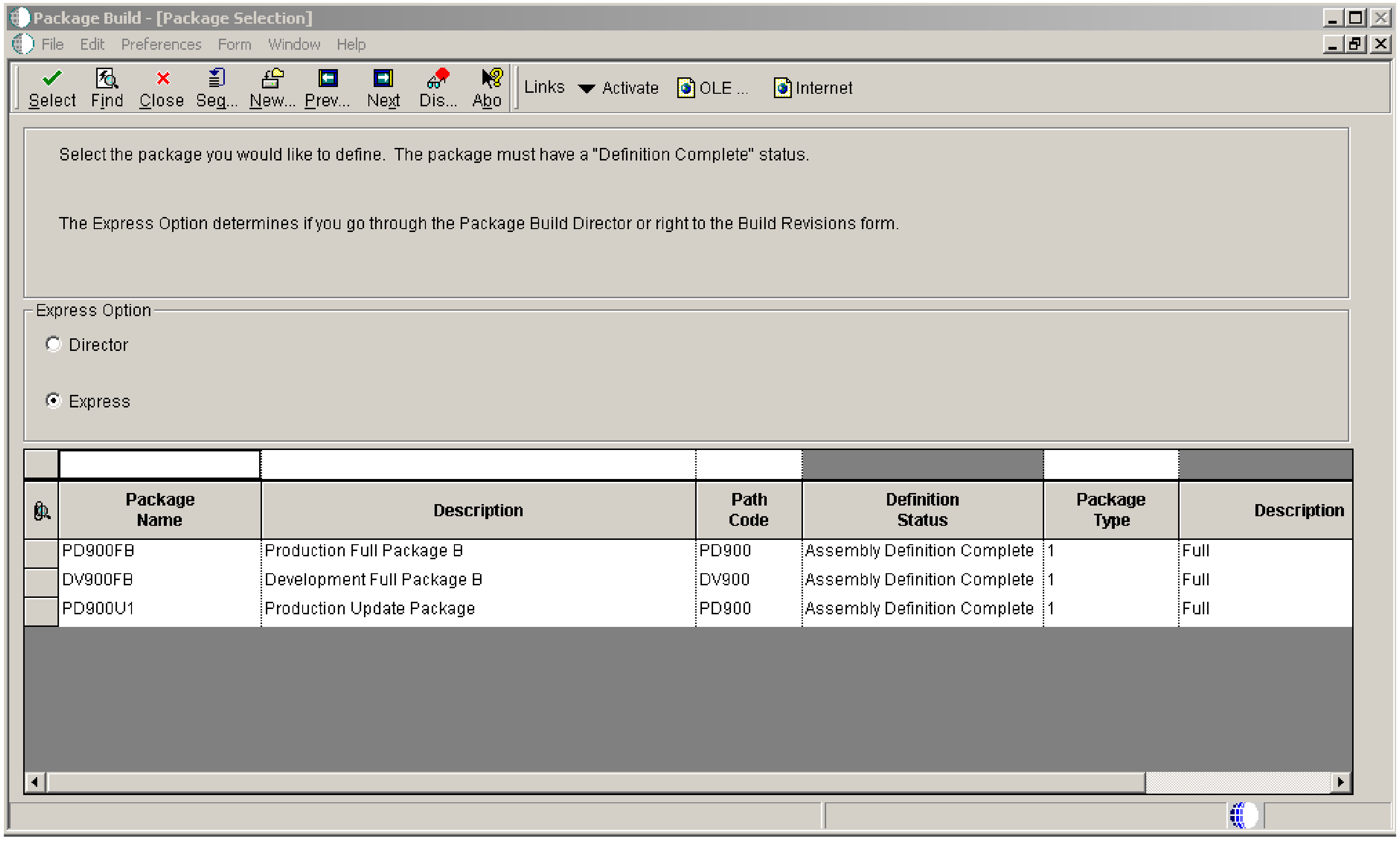
5. ESU build (Tools Release 9.2.6)
Mark this processing option if this is an ESU build.

6. Compress Application Component use in Server Manager (Tools Release 9.2.7)
Enter **1** if you want the Compress Application Component to be automatically selected. Leave this processing option blank or enter **0** if you do not want to create the Application Component file.

7. Build Net Changed Objects Only (Tools Release 9.2.7)
Enter **1** if you want to build a full package with only the business functions and tables that were changed. The specs tables will be fully built. This will shorten the build time. Leave this processing option blank or enter **0** if you want to build all objects in the full package.

Defining a Package Build

Access the Package Selection form.


Package Build - [Package Selection]
File Edit Preferences Form Window Help
Select Find Close Seq... New... Prev... Next Dis... Abo Links Activate OLE ... Internet

Select the package you would like to define. The package must have a "Definition Complete" status.

The Express Option determines if you go through the Package Build Director or right to the Build Revisions form.

Express Option
( ) Director
[✓ Express]

Package Name	Description	Path Code	Definition Status	Package Type	Description
PD900FB	Production Full Package B	PD900	Assembly Definition Complete	1	Full
DV900FB	Development Full Package B	DV900	Assembly Definition Complete	1	Full
PD900U1	Production Update Package	PD900	Assembly Definition Complete	1	Full

1. Find and select the defined package that you want to build.

2. If the package definition has a status of **In Definition,** you must change the status to **Assembly-Definition Complete** before you build the package. To change the status, select the package and select Activate from the Row menu.
3. On the Package Selection form, in the Express Option pane, select one of these options:

Option	Description
Director	Select this option if you want to configure the package build. Director enables you to navigate the package build definition forms.
Express	Select this option if you want to accept the default build parameters. Express enables you to accept the default options for the package build and skip the package build definition forms.

4. If you selected the Express option, skip to the Reviewing Package Selections task. If you selected the Director option, continue with the next task.
5. On the Package Build Location form, select one or both of these options:

Option	Description
Client	Select to indicate that the package is being built for installation on client workstations.
Server(s)	Select to indicate that the package is being built for installation on one or more servers.

6. If you are building a package for client workstations only, click Next and proceed to step 10.
7. If you are building a server package, you can specify the Shared Location for the shared spec database and click Next.

   Note: The default shared spec database is always the central objects data source for the package path code.

8. To select a server on the Server Selection form, and then double-click the row header for the server.

   A check mark indicates your selection. You can select multiple servers.

   Note: Servers are automatically selected for an update package. They are selected based on the server selection of the parent package.

9. Click Next.
10. On the Build Specification Options form, select Build Options to take the package definition and copy and convert objects from the central data source to the replicated format used by workstations.
11. Complete these fields and click Next:

Field	Description
All Specification Tables	Select this option if you want to build all specification tables into the package.

Field	Description
Individual Specification Tables	Select this option if you would like to select individual tables to include in the package. All of the tables listed on the Individual Specifications Selection form will be included in the package.
Stop Build Option	Indicate the point at which the system should stop the build. You can continue building on all errors, stop building on specification errors, stop building on business function errors, or avoid compressing when errors exist.
Replace jde.ini	For update packages, indicate if you want a new jde.ini file delivered with the package. Leave this unchecked unless the jde.ini file has changed. For example, the jde.ini may change when you perform upgrades or when you re-configure in release master.

12. If you chose to build individual specification tables, the Individual Specification Selection form appears.
13. To indicate that you do not want to build a specification table, clear its option.

    You can clear multiple options.
14. Click Next.

    For a full package or for an update package that includes business functions or tables with table event rules, the Business Function Options form appears.
15. Complete these fields and click Next:

Field	Description
Generate NER	Specify whether you want to generate the named event rules and table event rules into .c and .h files. This will generate n<*xxxxxx*>.h and n<*xxxxxx*>.c for the named event rules and f<*xxxxxx*>.hxx and f<*xxxxxx*>.c for table event rules.
Build Functions	Specify whether you want to build business functions. This will compile and link all the business functions, named event rules, and table event rules into a .dll to run in EnterpriseOne.
Generate NER Headers for Opposite bitness	Specify whether to generate the header files for the opposite bitness in which the package is being built. For example, if the package is building a 32-bit client or client/server package (system is 32-bit), and the path code has been converted to 64-bit, then when business functions are built on the client, the NER will be generated for the 64-bit header files. When the package is finished, there will be NER .h files in the include64 directory. These will be needed if a client package is installed and used for development work. This is automatically checked if Build Mode is Debug. If Build Mode is Optimize, it is not checked.
Build Mode	Specify the build mode, such as debug or optimize.

Field	Description
Stop-Build Option	Specify what action to take if errors occur while building business functions.
Build BSFN Documentation	Specify whether you want to build the documentation for the functions.
Clear Output Destination First	Indicate if you want the destination directory for the functions to be cleared before the build.
(Tools Release 9.2.7.3) Repair Repository Record	Select to repair any BSFN/TBLE/UBE/BSVW or APPL with files in the F98780R with a missing record or missing .c, .h or bitmap.
(Tools Release 9.2.9) Build Net Changed Objects Only	Select to build all specs and only business functions, tables, UBE with files, and business view with files that were changed since the last active package. This results in a shorter build time.

16. On the Compression Options form, select Compress Client Options if you would like to compress the client package.

    - Select this option to compress the applications included in the package, and to specify options for the compression process.

Note: (Prior to Tools Release 9.2.5) After applying JD Edwards EnterpriseOne Tools 9.2, the Data.CAB needs to be compressed for the first full package for any path code where you are building a client package with Tools 9.2. Failure to do this will result in errors during the client installation because it will not be able to decompress the Data.cab. On the Compression tab, for Compression Options, select "Compress Data" in order to compress the data.CAB. Also, if you did not get the Tools Release through Server Manager, select "Compress Foundation". This only needs to be checked for the first package for each path code.

17. If you are compressing the client package, select from these options:

Option	Description
All Client Directories	Select to compress all of the directories listed on the Individual Directory Selection form.
Individual Client Directories	Select to compress only certain directories which you specify.
Compress Data (Prior to Tools Release 9.2.5)	Indicate whether to compress the data in a package after the package is created. Compress Data compresses the Supported Local Database that is associated with this package.
Compress Foundation	Indicate whether to compress the foundation files in the package after the package is created. Compress Foundation compresses the foundation that is associated with the package.

18. Select Compress Server Options if you would like to compress the server package.
    - Select this option to compress the applications included in the package, and to specify options for the compression process.

      You should select Compress Server Options if you plan to build the package on one enterprise server and deploy it to another enterprise server.
19. If you are compressing the server package, select from these options:

Option	Description
All Server Directories	Select to compress all of the directories listed on the Individual Directory Selection form.
Individual Server Directories	Select to compress only certain directories which you specify.
(Tools Release 9.2.7) Compress Application Component	Select to compress the application components (which includes the **`bin64`** and **`spec`** directories) into a jar file. This jar file can then be used to upload to the Server Manager and distribute to an Enterprise Server. You can create a new Enterprise Server Instance using this application component along with a System Tools release.

20. Click Next.

    If you chose to compress individual directories, the Individual Directory Selection form appears.
21. On the Individual Directory Selection form, indicate that you want to compress a directory for the client or server by clicking its option to select it and click Next.

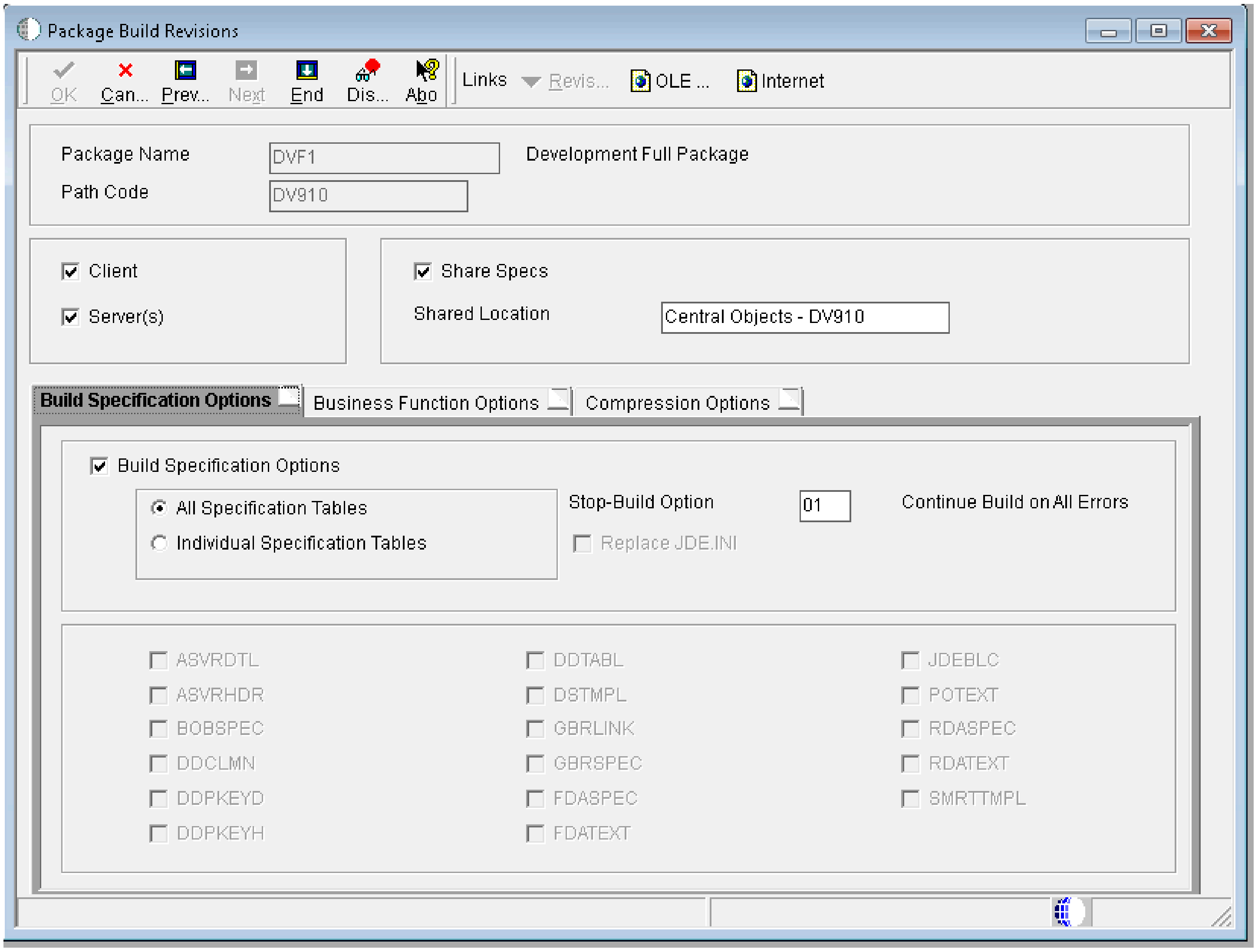
    You can select multiple options.
22. If the package does not include features, skip to the next task.
23. On the Build Features form, if you want to build a feature.inf file with the package, select the Build Feature INFs option.

    When you select this option, the Compress and Build options become available.

    See *Configuring Features During the Package Build Definition*.
24. Click Next.
25. Review the package build selections and click End.

Reviewing Package Build Selections

Access the Package Build Revisions form.



Package Name: DVF1 Development Full Package
Path Code: DV910

[✓] Client
[✓] Server(s)

[✓] Share Specs
Shared Location: Central Objects - DV910

Build Specification Options | Business Function Options | Compression Options

[✓] Build Specification Options

(●) All Specification Tables
( ) Individual Specification Tables

Stop-Build Option: 01 Continue Build on All Errors
[ ] Replace JDE.INI

[ ] ASVRDTL	[ ] DDTABL	[ ] JDEBLC
[ ] ASVRHDR	[ ] DSTMPL	[ ] POTEXT
[ ] BOBSPEC	[ ] GBRLINK	[ ] RDASPEC
[ ] DDCLMN	[ ] GBRSPEC	[ ] RDATEXT
[ ] DDPKEYD	[ ] FDASPEC	[ ] SMRTTMPL
[ ] DDPKEYH	[ ] FDATEXT	

1. Review current build options, business function options, compression options, and feature options that you specified for the package.
2. Click the tab for the type of option that you want to change, and make any changes.

   Only tabs for options that you selected appear on this form.

   See *Defining a Package Build*.
3. When you are finished reviewing or changing the build options, click End to exit the Package Build Definition Director, or click OK to accept changes to an existing package.
4. On Work With Package Build Definition, activate the package by choosing Active/Inactive from the Row menu.

   After you enter the build options for a package, you can easily revise any of those options using the Package Build Revision form. You do not need to go through all of the forms in the Package Build Definition Director to revise build options.

Building a Package

Access the Work With Package Build Definition form.

1. Select Active/Inactive from the Row menu to activate the package.
2. Select Submit Build from the Row menu when you are ready to initiate the package build.
3. Select one of these options and click OK.
    - On Screen
    - To Printer

      The form closes and the system begins building the package. Build time varies, depending on the number and size of the items in the package. A build could take five minutes for a small package, or several hours for a full package that contains all applications. When the build finishes, the report either appears on the screen or prints, depending on the destination that you specified.
4. Review the report to make sure that all components in the package were built successfully.

   If the report indicates any errors, review the error logs for more detail.

   If the package build finishes successfully, you can schedule the package for deployment.

Building a Package From the Web (Release 9.2.5)

Starting with Tools Release 9.2.5.0, a web application is available to define a package.

1. Access the Web application through P9621W or the Package Assembly application.
2. Select Define Build from the Row exit. This will take you directly to the Package Build Revision (P9621W) application.
3. Select Ok.
4. Select the servers to build on and then select OK.
5. Again, select Ok.
6. Activate this package. Select Submit from the Row exit build to build the package.

To check the status of the package, use Batch Version (BV fastpath). Go to Submitted jobs and select R9621S. When the package is complete, the status is set to Done for R9621S and R9622C. You can also monitor the package on the server in the package directory or the deployment server under packages\<package name>\serverLogs.

(Release 9.2.9.3) Accessing Package Build Logs and Reports Using the Work With Package Build (P9621W) Web Application

Starting with Tools Release 9.2.9.3, you can download package build logs and access the **Server Package Build** (R9621S) and **Client Package Build** (R9622C) reports using the **Work With Package Build** (P9621W) web application.

Downloading Log Files Using the Work With Package Build Web Application

When a package is built from the web interface for the Enterprise Servers and Development Client, you can check the status of the build and download the server and client logs anytime during and after the package build.

Select the package in the **Work With Package Build** web application and click the View Log icon in the **Package Build Detail** section of the form to download the package logs.

The system retrieves package logs from the enterprise server and deployment server, contains them in a compressed file, and then imports it into the **EnterpriseOne Software Archive Repository table** (*F96700R*). The compressed log file is available even after deployment of the package and is deleted from the F96700R table when the package is deleted.

Viewing Package Build Reports Using the Work With Package Build Web Application

After the package build is complete, select the package in the **Work With Package Build** web application and click the View Output icon corresponding to the server or client, in the **Package Build Detail** section of the form to view the **Server Package Build** (R9621S) and **Client Package Build** (R9622C) reports.

Note: *Click here to view an OBE of this feature.*

Incorporating Features into Packages

This section provides overviews of the feature build and deployment process and the JD Edwards EnterpriseOne Feature Based Deployment Director and discusses how to:

- Create a feature.
- Define a file set.
- Define a registry setting.
- Define a shortcut.
- Define additional package build processes.
- Define additional install processes.
- Define an initialization file.
- Define a new open database connectivity (ODBC) data source.
- Import an existing ODBC data source.
- Review feature components.
- Copy features.
- Add a feature to a package.
- Configure features during the package build definition.
- Configure features for an existing package build definition.

Understanding the Feature Build and Deployment Process

A feature is a set of files or configuration options, such as registry settings, that is copied to a workstation or server to support an application or other functions. Like objects, features are built into a package and deployed to the workstations and servers that require the feature components.

Note: Oracle's JD Edwards EnterpriseOne web development clients require a specific feature component to develop web-based objects.

Note: Oracle's JD Edwards EnterpriseOne machines that run Microsoft Windows, use business functions built with a Microsoft Visual C++ 2008 or higher level compiler, and do not have the same level Microsoft Visual C++ compiler installed locally, require a specific feature that installs the appropriate Microsoft Visual C++ Compiler runtime libraries.

See *"Appendix: Using the Microsoft Visual C++ Compiler" in the JD Edwards EnterpriseOne Administration Guide*

You might also want to include any of these features when you build a package:

- ActiveX controls.

  The Application Design Aid tool enables you to include ActiveX controls in applications. If ActiveX controls are delivered with the software, you need a way to copy these controls to the workstation.
- Open Data Access (ODA) data sources.

  ODA requires that additional ODBC data sources be created on any workstation or server that uses ODA.
- Sales Force Automation databases.

  The Sales Force Automation feature requires that you install a separate Supported Local Database on the workstation so that it can be disconnected from the network during offline operation. You must also write a registry setting that indicates that the machine is used offline.
- GenCorba, GenCom, and other third-party interfaces or products.

  Each of these products and interfaces requires additional components on the workstation and server in order to function. As functionality expands to support additional third-party products and interfaces, these products will each have their own set of supporting files.

For software releases prior to JD Edwards EnterpriseOne 8.10, custom programming was required to add feature components to the workstation and server. You can now use familiar tools such as the JD Edwards EnterpriseOne Package Assembly Director and Package Build Definition Director to create a package that contains the feature, and then you can deploy it using the JD Edwards EnterpriseOne Package Deployment Director or multitier deployment.

Because feature components are not objects, the process for incorporating feature components into a package is slightly different from the normal package build process. Specifically, you must first define the feature before you can add it to a package.

Feature Definition

Before adding the feature to a package, you must first define it using Oracle's JD Edwards EnterpriseOne Feature Based Deployment Director. During feature definition, you specify the feature name and type, enter a brief description, and specify installation parameters.

The forms in the JD Edwards EnterpriseOne Feature Based Deployment Director enable you to:

- Create a file set.
- Define registry settings.
- Define a Microsoft Windows shortcut.
- Enter initialization file information.
- Add ODBC data sources.
- Specify the feature build sequence.
- Enter information for third-party products.

Feature Selection During Package Assembly

After you have defined the feature, it is ready to be included in a package. Use the Package Assembly Director to assemble the package as you would any other package. When you assemble the package, feature-specific forms enable you to specify the features that you want to include.

Feature Configuration During Package Build Definition

After you have assembled the package that contains the features, you can use the Package Build Definition Director to define the build for the package. Forms in this director enable you to select the file sets that will be compressed within the package, and to specify the processes that will be run before and after the feature is built.

Package Deployment

After you have built the package, you are ready to schedule it for deployment by using Oracle's JD Edwards EnterpriseOne Package Deployment Director. The procedure is the same as the procedure that you use to schedule packages that do not include features.

Workstation Installation and Deployment Server Installation

After you have deployed the package to workstations and deployment servers, use Oracle's JD Edwards EnterpriseOne Workstation Installation and Deployment Server Installation applications to install the package.

Feature Entries in the Package.inf File

When a package contains a feature, the package.inf file [Features] section provides the feature name and the location of the feature.inf file that the system creates for each feature. The feature.inf file contains information pertaining to the feature, such as shortcut information, registry settings, initialization file settings, and environment information.

Installation of Packages Containing Features

You install packages containing features on workstations and servers in the same way in which you install any other package: through the JD Edwards EnterpriseOne Workstation Installation and Deployment Server Installation applications.

When you launch either of these installations, you can select the Custom option to select the features that you want to install.

Note:

- *Features*.
- *Adding Features to a Package*.
- *Workstation Packages*.
- *JD Edwards EnterpriseOne Application Release 9.0 Installation Guide*.

Understanding the Feature Based Deployment Director

The JD Edwards EnterpriseOne Feature Based Deployment Director enables you to define the feature so that it can be included in a package and then deployed to workstations and servers. The forms in the director enable you to specify the name and type of the feature, as well as the different feature components. The Feature Information form enables you to select the types of components to include in the feature, and determines the subsequent forms that appear in the JD Edwards EnterpriseOne Feature Based Deployment Director.

For this release, the Platform value must always be **80** for CLIENT. Future releases will enable you to select alternative platforms.

Throughout the feature definition process, you can always proceed to the next or previous form by clicking Next or Previous. Also, regardless of where you are in the process, you can always cancel the feature definition by clicking Cancel.

Copying a Feature Definition

The JD Edwards EnterpriseOne Feature Based Deployment Director includes a copy function that enables you to copy an existing feature and rename it as a new feature. This feature is especially useful if you want to create a feature definition that closely matches an existing feature definition.

Forms Used to Incorporate Features into Packages

Form Name	FormID	Navigation	Usage
Feature Information	W9326C	Package and Deployment Tools (GH9083), Package Assembly Select Features from the Form menu. Click Add. Click Next.	Define a feature and add one or more components to the feature.
File Set Definition	W9326J	From the Feature Information form, select File Set and click Next.	Enter information about any file sets that must be installed on the workstation or server for the feature to function properly. A file set is a collection of files that must be installed on the workstation or deployment server for the feature to function correctly.

Form Name	FormID	Navigation	Usage
Registry Definition	W9326D	From the Feature Information form, select Registry and click Next until the Registry Definition form appears.	Enter information that should be added to the Microsoft Windows registry as part of the feature installation. Registry information that you enter on this form will be delivered in the package that contains the feature.
Shortcut Definition	W9326G	From the Feature Information form, select Shortcut and click Next until the Shortcut Definition form appears.	Use this form to add a shortcut for the feature to the Windows desktop. The system creates a shortcut on the desktop after the feature is installed.
Shortcut Advanced Options	W9326P	From the Shortcut Definition form, select Advanced from the Form menu.	Enter advanced shortcut options.
Additional Package Build Processes	W9326H	From the Feature Information form, select Additional Package Build Processes and click Next until the Additional Package Build Processes form appears.	Specify a batch application or executable program to run either before or after the package that contains the feature is installed.
Additional Install Processes	W9326K	From the Feature Information form, select Additional Install Processes and click Next until the Additional Install Processes form appears.	Enter information about third-party applications that should be run when the package is installed.
Initialization File (INI) Definition	W9326I	From the Feature Information form, select Initialization Files (INI) and click Next until the Initialization File (INI) Definition form appears.	Enter information that should be written to an initialization file (such as jde.ini) as part of the feature installation. The INI file is automatically updated when the package is installed.
ODBC Data Source Definition	W9326N	From the Feature Information form, select ODBC Data Sources and click Next until the ODBC Data Source Definition form appears.	Enter information for any ODBC data sources that must be added to support the feature.

Form Name	FormID	Navigation	Usage
Local Data Sources	W9326O	On ODBC Data Source Definition, select Import from the Form menu.	Select previously created data sources that reside locally on your machine.
Features Summary	W9326L	From the Feature Information form, click Next and add the each feature component. After you add all the components, the wizard displays the Features Summary form.	Review and modify information that you entered on any of the Feature Based Deployments forms.
Feature Copy	W9326M	Package and Deployment Tools (GH9083), Package Assembly Select Features from the Form menu. Select the feature from which to copy the definition, and click Copy.	Copy an existing feature and rename it as a new feature. This function is useful if you want to create a feature definition that closely matches an existing feature definition.
Feature Component Selection	W9601AB	Package and Deployment Tools (GH9083), Package Assembly Click Add to create a new package. Enter the forms in the Package Assembly Directory until the Features Component form appears. Click Browse. Package and Deployment Tools (GH9083), Package Assembly Select a package and then select Package Revisions from the Row menu. On Package Component Revisions, click the Features button.	Add defined features to a new package. Add defined features to an existing package that is open for revision.

Form Name	FormID	Navigation	Usage
		To add a feature, click Browse.	
Build Features	W9621B	Package and Deployment Tools (GH9083), Package Build Click Add to launch the Package Build Definition Director. Click Next and complete the screens until you come to the Build Features form. Package and Deployment Tools menu (GH9083), Package Build Find and select the package that contains features. Select Build Revisions from the Row menu. Click the Build Features tab.	Enables you to specify whether the system builds feature INF files for the features in the package. If you defined a fileset component in the feature, you can select to compress it. If any additional package build processes are included in the feature, you must click Build Processes and select them before they will run during package build.

Creating a Feature

Access the Feature Information form and complete the following fields.

Feature
Enter a name for the feature.

Feature Type
Enter the feature type, if applicable.

Description
Enter a description of the feature.

Required
Select this option if the installation of this feature is mandatory for both Compact/Production and Typical/Development installs. Inclusion of this feature cannot be overridden when the package is installed.

Not Required
Select this option if the installation of this feature is optional. Whether the feature is installed depends on the options that you select (Compact/Production and Typical/Development). Inclusion of the feature can be overridden when the package is installed.

Compact/Production

Select this option if this feature is to be included in a Compact/Production install by default. This option can be overridden when the package is installed if Not Required is also selected.

Typical/Development
Select this option if this feature is to be included in a Typical/Development install by default. This option can be overridden when the package is installed if Not Required is also selected.

File Set
Select this option if the feature is a file set.

Registry
Select this option if the feature is a registry setting.

Shortcut
Select this option if the feature is a shortcut.

ODBC Data Sources
Select this option if the feature is an ODBC Data Source.

Additional Package Build Processes
Select this option if the feature is an additional process to be performed during the package build.

Additional Install Processes
Select this option if the feature is an additional process to be performed during the install process.

Initialization Files (INI)
Select this option if the feature is an initialization file.

Defining a File Set

Access the File Set Definition form and complete the following fields.

File Set
Enter comments or memoranda in this free-form text field.

File Set Description
Enter a description for the group of files.

Source Path
Enter the path that identifies the source location of the file set.

Compress
Select this option to compress the file.

Target Path
Enter a path to identify the target location of the file set.

The source path tells the system where to find the file set to be copied into the package, and the target path indicates the location to which the file set should be copied when the package is installed. Although a feature can have an unlimited number of file sets, each file set can have only one target path.

Note: You can also use this form to modify or delete any previously defined file sets. Existing file sets appear in the tree structure on the right side of the form. To modify a file set, select the file set on the tree structure and modify any of the fields for the file set. To delete a file set, select the file set and click Delete. Always select Save Node from the Form menu when you are finished adding file set information.

Defining a Registry Setting

Access the Registry Definition form and complete the following fields.

Registry
Enter an identifier for the registry modification.

Registry Root
Enter the root key in the registry.

Key
Enter the key for a registry value.

Name
Enter the registry value name.

Value
Enter the registry value.

Value Type
Enter the data type in which the value is stored in the registry.

Note: You can also use this form to modify or delete any previous registry definitions. Existing registry definitions appear in the tree structure on the right side of the form. To modify a registry definition, select the item on the tree structure and modify any of the fields for the registry definition. To delete a registry definition, select the item and click Delete. Always select Save Node from the Form menu when you are finished entering registry information.

Defining a Shortcut

To define a shortcut component, you enter a shortcut definition, and then you can enter advanced shortcut options.

Entering a Simple Shortcut Definition

Access the Shortcut Definition form and complete the following fields.

Shortcut
Enter a name that identifies a unique shortcut to a user's computer.

Name
Enter the name of the shortcut.

Target
Enter the path and file name of a target file.

Entering Advanced Shortcut Options

Access the Shortcut Advanced Options form and complete the following fields.

Arguments
Enter the parameters that are entered at the command line for the shortcut.

Description
Enter a description of the shortcut.

Hot Key
Enter a key sequence that, when pressed, automatically launches the shortcut.

Icon
Enter the path and name of the icon file, based on a relative target path.

Icon Index
Enter the icon index for a shortcut.

Show Command
Specify the size of the window after the shortcut is launched. For example, the window might be minimized or maximized.

Work Directory
Enter the identifier of the directory path or the working directory of a shortcut.

Defining Additional Package Build Processes

Access the Additional Package Build Processes form and complete the following fields.

Process Name
Enter the name of the build process.

Description
Enter a description of the build process.

Sequence
Enter a number to identify the order in which the process will be run relative to the other processes that run during the package build.

Synchronous Execution
Select this option to indicate whether the package build job waits for the process to finish before it continues.

Batch Application or Executable
Specify whether the process is an application or an executable.

UBE Name
Enter the name of the batch application. Only applies if batch application was selected.

UBE Version
Enter the version of the batch application. Only applies if batch application was selected.

Machine Name
Enter the name of the server or workstation on which the batch application will run. Only applies if batch application was selected.

Executable Name
Enter the name of the executable program that the system launches to install the third-party software. Only applies if executable program was selected.

Target Path
Enter the path and file name of a target file. Only applies if executable program was selected.

Parameters
Enter the executable parameters that the setup program uses to install the third-party software. Only applies if executable program was selected.

Note: You can also use this form to modify or delete any previously defined processes. Existing processes appear in the tree structure on the right side of the form. To modify a process definition, select the item on the tree structure and modify any of the fields for the definition. To delete a process definition, select the item and then select Delete or Delete Node After from the Form menu, depending on whether you want to delete a process that is executed before or after the feature is installed. You can run the process either before or after the feature is built. When you are finished adding process information, select either Save or Save Node After from the Form menu, depending on when you want the process to run.

Defining Additional Install Processes

Access the Additional Install Processes form and complete the following fields.

Third Party
Enter the name of the third-party component.

Description
Enter a description of third-party software.

Sequence
Enter a number to identify the order in which this process will run relative to the other additional install processes.

Execute After the Install
The JD Edwards EnterpriseOne client install will run the third-party process after the client has installed.

Executable Name
Enter the name of the program that launches the third-party software.

Target Path
Enter the path to the executable file. Do not include the name of the file.

Parameters
Enter the executable parameters that the system passes to the third-party program.

Note: Select Save from the Form menu when you finish adding third-party product information.

Defining an Initialization File

Access the Initialization File (INI) Definition form and complete the following fields.

Initialization INI
Enter the identifier of an initialization file component.

File Name
Enter the name of the initialization file.

Target Path
Enter the path of the INI file.

Section Name
Enter the name of the application section in an initialization file.

Key Name
Enter a key in the initialization file that is to be added, modified, or removed.

String
Enter the value of the key in an initialization file.

Option
Enter the option that identifies the action associated with the key in the initialization file.

> **Note:** You can use this form to modify or delete any previous initialization file definitions. Existing definitions appear in the tree structure on the right side of the form. To modify an initialization file definition, select the item in the tree structure and modify any of the fields for the definition. To delete an initialization file definition, select the item and click Delete. When you finish adding initialization information, select Save Node from the Form menu.

Defining a New ODBC Data Source

Access the ODBC Data Source Definition form and complete the following fields.

ODBC Data Source
Enter the name of the data source.

> **Note:** When you select Save Node from the Form menu, the system activates the Microsoft Windows control panel applet that displays the ODBC Data Source forms where you can enter the data source information.

Importing an Existing ODBC Data Source

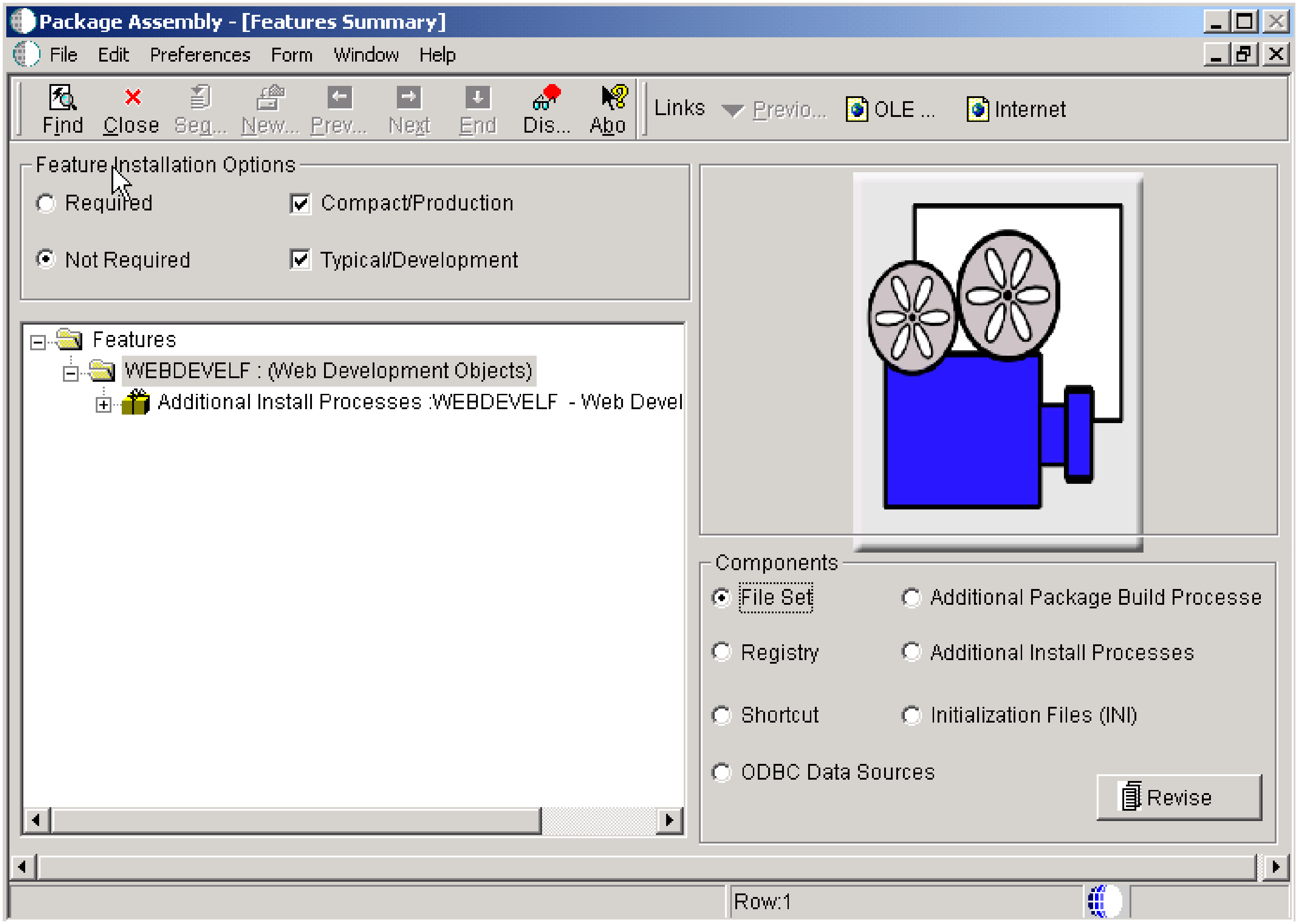
Access the Local Data Sources form.

1. Press the Ctrl or Shift key to select one or several data sources, and click Select to add the data sources to the feature.

   The ODBC Data Source Definition form reappears.
2. When you are finished adding data source information, select Save Node from the Form menu.
3. Click Next.
4. To modify existing data sources, enter the data source name and then select Modify from the Form menu. The ODBC Data Source Revisions form appears. Use this form to make changes to the data source.
5. When you are finished, click OK to return to the ODBC Data Source Definition form.

Reviewing Feature Components

Access the Features Summary form.



Feature Installation Options

[ ] Required [✓ Not Required]

[✓] Compact/Production

[✓] Typical/Development

Features
- WEBDEVELF : (Web Development Objects)
  - Additional Install Processes :WEBDEVELF - Web Devel

Components

[✓ File Set] Additional Package Build Processe
Registry
Additional Install Processes
Shortcut
Initialization Files (INI)
ODBC Data Sources

1. Select a component in the right pane and click the Revise button to review the information for that component.
2. If needed, change the field values for the selected component and click Save.

3. Repeat the previous steps to modify other components.
4. When you are finished defining the feature, click End.

Note:

- *Revising an Existing Package*.

Copying Features

Access the Feature Copy form.

1. Complete these fields:
    - Feature
    - Feature Type
    - Description
2. Select one of these options:

Option	**Description**
Required	The installation of this feature is mandatory for both Compact/Production and Typical/ Development installs. Inclusion of this feature cannot be overridden when the package is installed.
Not Required	The installation of this feature is optional. Whether the feature is installed depends on the options that you select (Compact/Production and Typical/Development). Inclusion of the feature can be overridden when the package is installed.

3. Select one or both of the options that follow.

    If you chose Required, both of these options are automatically selected.

    - Compact/Production

        When selected, this feature is included in a Compact/Production install by default. This option can be overridden when the package is installed if Not Required is also selected.
    - Typical/Development

        When selected, this feature is included in a Typical/Development install by default. This option can be overridden when the package is installed if Not Required is also selected.
4. Click OK.
5. To revise the new feature definition, select the feature and select Revise Feature from the Form menu.

Adding a Feature to a Package

Access the Feature Component Selection form.

1. Click Find to display the list of available features.

   > **Note:** Before a feature is available for inclusion in the package, you must first define the feature.

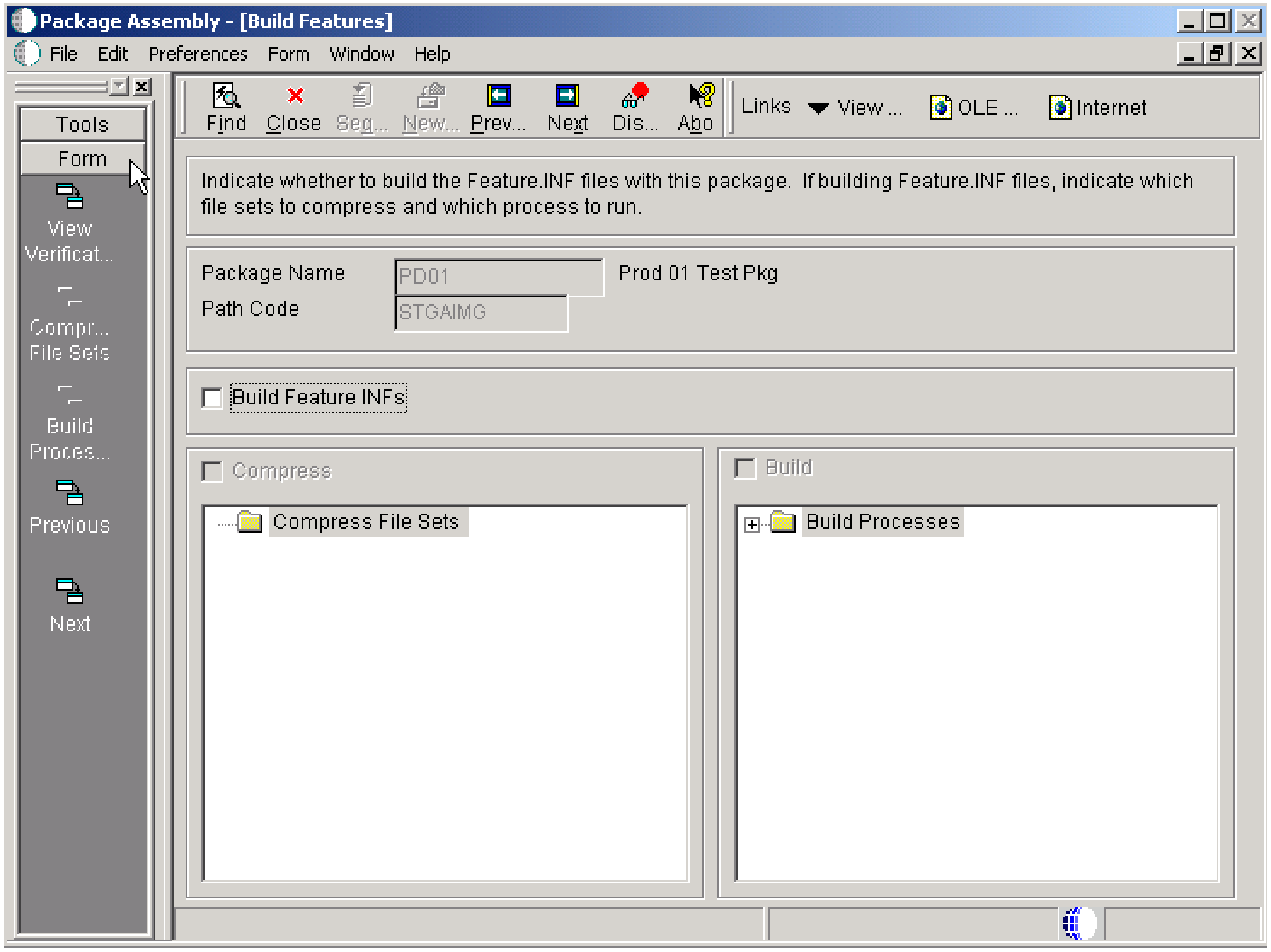
2. Use one of these methods to select one or more features to include in the package:
   - Select a feature and click the Select button.

     (Press the Ctrl or Shift key to select multiple features.)
   - Double-click each feature.
3. When you are finished adding features, click Close to return to the Features Component form. The selected features appear.
4. Click Next and complete the remaining forms to finish assembling the package.

   > **Note:** To delete a feature that was previously included in the package, on the Features Component form select the feature and then click Delete.

Configuring Features During the Package Build Definition

Access the Build Features form.


Package Assembly - [Build Features]
File Edit Preferences Form Window Help
Tools
Form
View Verificat...
Compr... File Sets
Build Proces...
Previous
Next
Find Close Seq... New... Prev... Next Dis... Abo
Links View ... OLE ... Internet

Indicate whether to build the Feature.INF files with this package. If building Feature.INF files, indicate which file sets to compress and which process to run.

Package Name: PD01 Prod 01 Test Pkg
Path Code: STGAIMG

☐ Build Feature INFs

☐ Compress
Compress File Sets

☐ Build
Build Processes

1. If you want to build a feature.inf file with the package, select Build Feature INFs.

   When you select this option, the Compress and Build fields become available if file sets or additional package build process components are included in the package.

2. Continue with one or both of these tasks:
   - To compress file sets
   - To build processes
1. Select Compress, and then select Compress File Sets from the Form menu.
2. On the File Set Selection form, select each feature that you want to include by choosing a file set and clicking Select.
3. When you are finished selecting file sets, click Close.
4. Continue either by performing the next steps, or by clicking Next and completing the remaining forms to finish defining the package build.
5. To build processes, select Build, and then click Select Build Processes.
6. On the Build Processes Selection form, select each process that you want to build by choosing a process and clicking Select.
7. When you are finished selecting processes to build, click Close.
8. From the Form menu, select Build Processes and manually select each process to run during the package build.

   You must complete this step or none of the processes will run, even though they are included in the feature.
9. Click Next and complete the remaining forms to finish defining the package build.

Configuring Features for an Existing Package Build Definition

Access the Build Features form.

1. Modify or add to any of these existing build feature settings:
   - Build Feature INFs
   - Compress
   - Build
2. If you select Compress, select Revise File Sets from the Form menu to modify file sets.
3. When you are finished modifying file sets, click Close.
4. If you chose Build, click Revise Processes to modify processes.
5. When you are finished modifying processes, click Close.
6. If you selected Build, from the Form menu, select Build Processes and manually select each process to run during package build.

   You must complete this step or none of the processes will run, even though they are included in the feature.
7. Click OK to complete the package build definition.

Viewing Package Build Records and Resubmitting Builds

This section provides overviews of package build history and the build status and discusses how to:

- View the package build history.
- View log files.
- Resubmit a package build.

- Change the build status.
- Reset the specification build and package build statuses.

Understanding Package Build History

The JD Edwards EnterpriseOne Package Build History program (P9622) enables you to view information pertaining to the build process, including the options and objects that you specified when you created the build definition. This program provides this build information:

- Package name.
- Path code.
- Date and time built.
- Name of the server for which the package was built.
- Current build status and status description.
- Current status of selected specification tables.
- Number of specifications written.
- Package records written and read.

The View Logs option on the Form menu enables you to view logs that contain additional information about the build process. Refer to these logs in the event that the build does not finish successfully and you need to review the errors that occurred during the build.

If a build does not finish successfully, you can use the Resubmit Build option to resume the build from the point at which the process stopped. Only the business functions and objects that did not build successfully will be built; the entire package will not be rebuilt.

In some cases, if a build is interrupted or otherwise unable to finish, you might need to reset the build status from Build Started to Build Definition Complete. Unlike the Resume Build feature, which continues the build from the point at which it failed, resetting the status enables you to start the build process from the beginning.

F96225 Table

The system maintains a history of the package build in the F96225 table. This table contains details about the package build statuses of any package components.

If you encountered errors during the build process and the package failed to build successfully, you can resubmit the package and continue building from the point at which the build failed. In this situation, the system reviews the F96225 table and rebuilds only the business functions or other package components that have a status of Not Built or Error. It does not build the entire package. This feature can save you a tremendous amount of time, especially if only a few package components failed to build successfully.

If you originally specified package compression, when you resubmit the package to resume building, the system automatically compresses the directories after it successfully builds the package.

Understanding the Build Status

In some cases, you might need to rebuild the package rather than resume the build from the point at which the build failed. Before you can do so, you must change the status of the package build from **Build Started** to **Build Definition Complete.**

When you reset the status of the package build, you can reset the status for the server only or for all servers and client workstations for which you want to build the package.

Forms Used to View Package Build History and Logs

Form Name	FormID	Navigation	Usage
Work With Package Build History	W9622A	Package and Deployment Tools (GH9083), Package Build History	Display information about the current build status and build options for selected computers.
View Logs	W9622B	Package and Deployment Tools (GH9083), Package Build History Select View Logs from the Form menu.	Check logs for errors that occurred during the build process.
Work with Package Build Definition	W9621L	Package and Deployment Tools (GH9083), Package Build W9621L	Change the package build status.
Reset Build Status	W9622C	From Work with Package Build History, find the package for which you want to reset the statuses, expand the package, and select an individual item. Select Reset Status from the Row menu.	Reset the spec status and pack status for a package to the statuses that you specify.

Viewing the Package Build History

Access the Work With Package Build History form.

1. Select CLIENT or the server or the spec data source to display information about the current build status for those computers.

   You can also expand the tree to view this information:

   - Build specification options
   - Compression options
   - Business function options
   - Objects

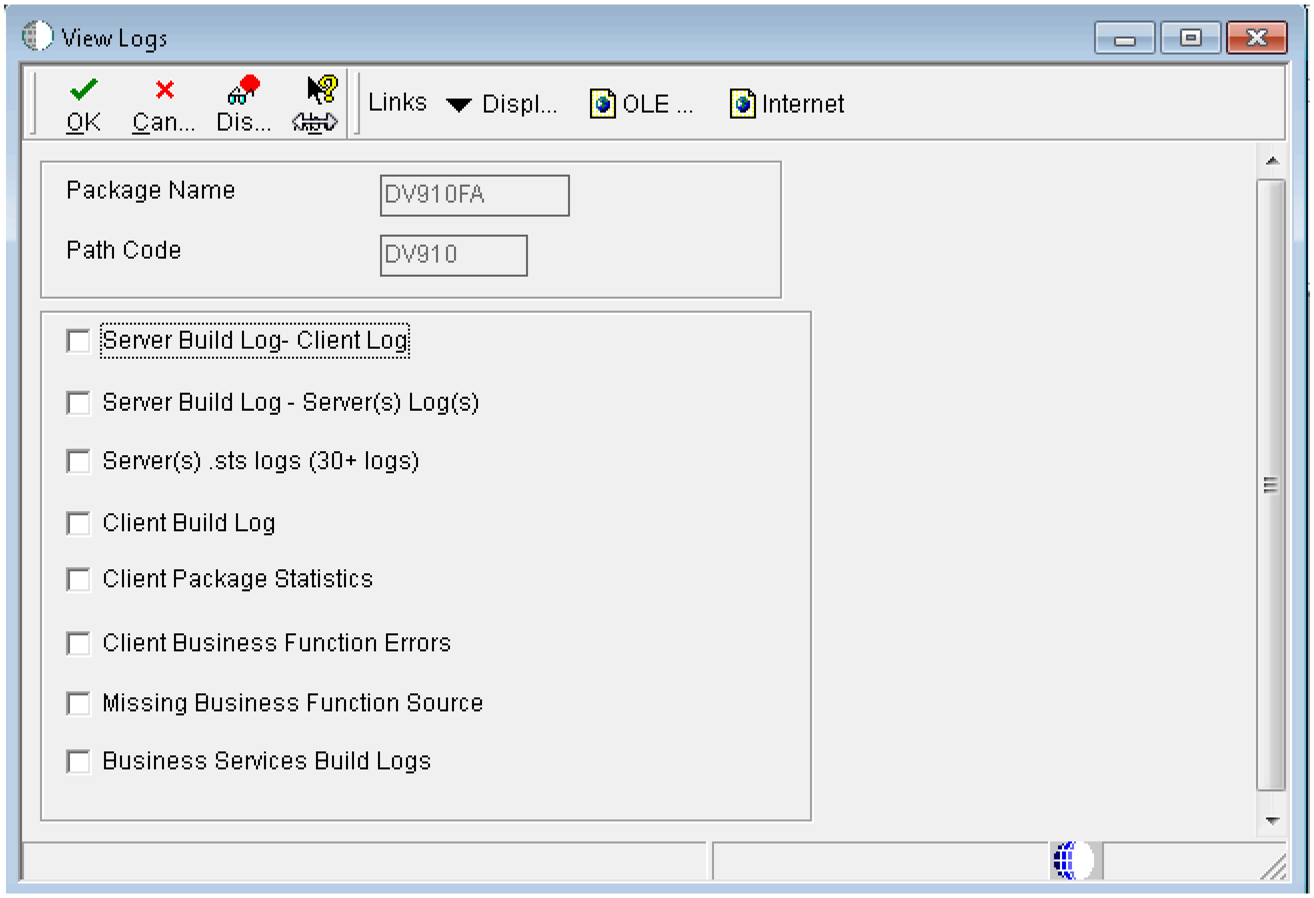
     These options and objects are those that you specified when you created the build definition for the package. For example, if you chose to build only selected specifications, you can determine the status for each specification, as well as other pertinent information.

     Note: Only specification options are built in the spec data source.

2. When you are finished viewing build history information, click Close.

Viewing Log Files

Access the View Logs form.



Package Name	DV910FA
Path Code	DV910

- [ ] Server Build Log- Client Log
- [ ] Server Build Log - Server(s) Log(s)
- [ ] Server(s) .sts logs (30+ logs)
- [ ] Client Build Log
- [ ] Client Package Statistics
- [ ] Client Business Function Errors
- [ ] Missing Business Function Source
- [ ] Business Services Build Logs

Server Build Log - Client Log
Select this option to view client-side processing.

The SvrPkgBuild.log is located directly under the <package name> directory.

Server Build Log - Server(s) Log(s)
Select this option to view server-side processing.

The <server name>_SvrPkgBuild.log files are located in the <package name>\serverlogs\<server name> directory.

Server(s) .sts logs (30+ logs)
These are the status files for all of the dlls that were built on the enterprise server.

The <dll name>.sts files are located under the <package name>\serverlogs\<server name> directory.

Client Build Log
The client package build log lists the steps completed in building the client package, as well as any errors that occurred during the process. This log is created for a client-only or client/server package.

The ClientPkgBuild.log file is located in the <package name> directory on the deployment server.

Client Package Statistics
This report contains the size of the directories on the client, number of files in each directory, and the size of the local package database.

The BuildReport.txt file is located under the <package name>\work directory.

Client Business Function Errors

Select this option to view any errors that occurred during the business function build for this package. Both errors and warnings display in this report. A summary appears at the end of the report that indicates how many errors and warnings occurred for each dll. Use this information to determine if a rebuild is necessary.

Business functions that appear on this report might be business functions that are still in development and have not yet been checked in. Business functions that have never been checked in do not have source, and therefore, are listed in the missing business function source errors log.

The BuildLog.txt file is located under the <package name>\work directory.

Missing Business Function Source
Select this option to see a list of all .c files that were missing when the business function was created. The program attempted to find these members because each had a record in the F9860 table. However, a matching source could not be found in the source directory. To resolve these errors, either delete the Object Librarian record or provide a source member.

The NoSource.txt file is located under the <package name>\work directory.

Business Services Build Log
Select this option to be able to view the results of the business services build for this package.

These logs are located under the <package name>\work\sbf directory.

Where to Find the Error Logs

To review error logs without using Oracle's JD Edwards EnterpriseOne Package Build History program (P9622), locate the desired log in the correct directory.

You can view the error logs by accessing the appropriate directory and opening the log with Microsoft Notepad or a similar application that enables you to display text files.

Resubmitting a Package Build

Access the Work With Package Build History form.

1. Select one of these options to find the package you want to resubmit:
   - Select a specific server to resubmit only the builds for that server.
   - Select the CLIENT heading to resubmit only the workstation builds.
2. Select Resubmit Build from the Row menu.

   If you generated NERs when you initially submitted the build, the system displays a window that asks whether you want to regenerate the NERs.
3. Click OK to regenerate NERs, or click Cancel to skip this process.

> **Note:** If you do not want to regenerate NERs, you can prevent this window from appearing by entering **2** in the Generate NER processing option for the Package Build History program.

4. Select one of these destinations for the build report, and click OK:

    - On Screen
    - To Printer

      The form closes, and the system begins to build the package. Build time varies, depending on the number and size of the items in the package. When the build is finished, the report either appears on the screen or prints, depending on the destination you specified.
5. Review the report to verify that the system successfully generated all components in the package.

   If the report indicates any errors, review the error logs for more detail.

Note:
- *Deploying Packages*.

Changing the Build Status

Access the Work with Package Build Definition form.

1. Find the package for which you want to reset the status.

   Below the package name, select the server or servers and client workstation for which you want to build the package.
2. From the Row menu, select Advanced.
3. On the Advanced Revisions form, click Reset to change the status of the package build from **Build Started** to **Build Definition Complete.**
4. Click OK.
5. If desired, select the package name and select Submit Build from the Row menu.
6. The program asks whether you want to delete the current build or to continue without deleting it; select one.

Resetting the Specification Build and Package Build Statuses

Access the Reset Build Status form.

1. Enter the desired statuses in the Spec Build Status and Pack Build Status fields.

   Both of these fields have a visual assist feature to help you determine the available statuses.

   Note: The values of these two fields are dependent on each other. If you change one value, be sure you understand the dependency on the other value.

2. Click Reset.
3. Click OK.

7 Deploying Packages

Understanding Package Deployment

After you assemble and build a package, you can select from several methods of deploying the package to workstations and servers throughout the enterprise. For workstations, the method that you select depends on whether Oracle's JD Edwards EnterpriseOne is already installed on the workstation.

> **Note:** After a new client package that was built with Microsoft Visual C++ Compiler "X" is deployed to a workstation, the login will fail from that workstation if the associated runtime libraries are absent. The errors in the log will include, "Business function library load failed." The same error can occur with a server package if the server package is built on a machine with an "X" compiler and then deployed to a server without a compiler. A specific feature that installs the appropriate Microsoft Visual C++ compiler runtime libraries is required. For Tools Release 9.2.3 and above, Visual Studio 2017 runtime redistributables are required.

See *"Appendix: Using the Microsoft Visual C++ Compiler" in the JD Edwards EnterpriseOne Administration Guide*

This section discusses:

- Deploying to workstations without JD Edwards EnterpriseOne.
- Deploying to workstations with JD Edwards EnterpriseOne already installed.
- Deploying to servers.
- Deploying to tiered locations.
- Deploying to workstations from CD.

Deploying to Workstations Without JD Edwards EnterpriseOne

If JD Edwards EnterpriseOne is not currently installed on a workstation, you can deploy the package through Oracle's JD Edwards EnterpriseOne Workstation Installation program. You can use JD Edwards EnterpriseOne Workstation Installation to deploy full packages, but you cannot use JD Edwards EnterpriseOne Workstation Installation to deploy an update package to a workstation on which JD Edwards EnterpriseOne is not installed.

JD Edwards EnterpriseOne Workstation Installation retrieves items that are specified in the package. A package is like a bill of materials with instructions that describe from where the system retrieves all of the necessary components that the JD Edwards EnterpriseOne Workstation Installation program deploys to the local workstation.

Deploying to Workstations with JD Edwards EnterpriseOne Already Installed

To reload a new package on workstations on which JD Edwards EnterpriseOne is already installed, use one of two methods:

- JD Edwards EnterpriseOne Workstation Installation (for full packages).
- Oracle's JD Edwards EnterpriseOne Deployment Director (P9631/P9631W) (for full and update packages).

After you assemble and build a package, use JD Edwards EnterpriseOne Deployment Director to schedule the package for deployment to individual workstations or to selected groups. On the specified deployment date, when the users who are scheduled to receive the package sign in, they are given the opportunity to load the package.

JD Edwards EnterpriseOne Deployment Director requires that JD Edwards EnterpriseOne be already loaded on the workstation. You can schedule a new full package to replace the existing package, or an update package to be merged with the existing package on the workstation.

Both deployment methods have advantages. JD Edwards EnterpriseOne Workstation Installation is a good method to use when you want to install a package immediately or soon after it is built, without having to schedule the package. Alternatively, JD Edwards EnterpriseOne Deployment Director is useful if you need to control when the package becomes available, if you want to make the package installation mandatory, or if you want to deploy the package to servers as well as to workstations.

Deploying to Servers

Servers receive the same package that you build for the workstation, but in a different format. When you assemble the package and create the package build definition, you can specify the servers to which you want to build and deploy the package. To deploy the package, you use the JD Edwards EnterpriseOne Deployment Director application (P9631/P9631W), which uses the same scheduling mechanism to deploy packages to workstations. In fact, you can easily schedule deployment to both client workstations and servers on the same form.

If you are deploying an update package that contains only UBEs, otherwise known as batch applications, the system marks the package as an "UBE-only" package. The system deploys a UBE-only package immediately, rather than waiting for the EnterpriseOne HTML server and waiting for all UBEs to finish. When the package is deployed, the system checks to see if the UBE in the package is currently being processed. If so, a lock is placed on that individual UBE so that the deployment can update the specifications. If not, the package deployment moves forward. This happens automatically and provides a faster deployment time for packages that contain only UBEs.

If you are deploying an update package that contains only APPs (interactive applications), the system marks the package as an "APP-only" package. The system deploys the APP-only package, waits for the EnterpriseOne HTML server (for one minute), and then deploys it immediately without waiting for any UBEs to finish.

If you are deploying an update package that contains only UBEs and APPs, the system marks the package as an "UBE/APPs only" package. In this case, the system waits for the EnterpriseOne HTML server (for one minute), and then follows the same logic as the UBE-only deployment.

All three of these scenarios happen automatically and provide a faster deployment time for packages that contain only UBEs, only APPS, or only UBEs and APPs.

Beginning with Release 9.2.7.0, table conversions are run after the deployment of the server package. For a full package build, the TC UBE runs all tables listed in the F98405 table. For update package, table conversions run only if tables are included at objects. For update packages, table conversion runs only if objects in the package are tables.

Deploying to Tiered Locations

Multitier deployment enables you to install software on workstations from more than one deployment location and more than one deployment machine. Use this deployment method if your site has more than 50 workstations performing software installations per day, or when workstation installations over your wide area network (WAN) are too slow.

Deploying to Workstations from CD

If your system has a CD writer, you can define the CD writer as a deployment location. Essentially, you define the CD writer as a pseudo deployment server from which you can copy a package onto a blank CD. You can then use this CD to install the software on workstations by using the JD Edwards EnterpriseOne Workstation Installation program that is included on the CD.

Defining Deployment Parameters

This section provides an overview of deployment parameters, provides prerequisites, and discusses how to:

- Define machines.
- Define locations.
- Define package deployment groups.
- Revise package deployment groups.

Understanding Deployment Parameters

Before you deploy packages, you must identify the workstations, servers, groups, or locations that will receive the package. Identifying these ensures that, when you are ready to schedule packages using the JD Edwards EnterpriseOne Deployment Director, the machines, groups, or locations that you want to receive the package will be available as package recipients.

A deployment group is a group of workstations that are classified by a criterion such as job function, team, or any other grouping that you specify. For example, you might have a software development group, a testing group, a production group, and so on. Oracle's JD Edwards EnterpriseOne Package Deployment Groups Revisions program (P9652A) enables you to define or revise groups that include several workstations.

A location is a group of workstations and servers that corresponds to a physical location. For example, you might have locations for Corporate and Branch, or for Building 5 and Building 7. Locations are also useful if you use multitier deployment or deploy across a WAN. In this case, you might define a location for each of your geographic locations. The JD Edwards EnterpriseOne Deployment Locations Application program (P9654A) enables you to define or revise machines and locations in your enterprise.

Both of these applications simplify the deployment process when you need to deploy a package to several users. Rather than requiring you to schedule deployment to each workstation or server, you can schedule deployment according to location or group.

When you enter a machine definition, you are really defining its usage in the configuration. For example, you can use a deployment server as a data server. When you enter machine definitions, consider these recommendations:

- An HTML server can be defined only as an HTML server, not as a data server, enterprise server, and so on.
- A deployment server should not be used as a workstation.
- A deployment server can be used as a data server.

- A deployment server should not be used as an enterprise server for tuning and performance reasons.

Locations

In some cases, an enterprise might span several buildings, cities, or countries. In these situations, you might deploy a package to a location rather than to individual workstations and servers. Then, a secondary deployment server at each location can deploy the package to the workstations and servers at that location.

The larger your enterprise, the more you can benefit from creating and deploying to locations. If you use multitier deployment to deploy packages to remote locations, the concept of locations is crucial.

In JD Edwards EnterpriseOne, a *location* is essentially a user-defined group of machines, databases, and environments. In some cases, the location is an actual physical location that is connected by a WAN, such as when you have remote offices that are geographically separate from your main office. For example, a location might be a floor in your office building, a separate building on the corporate campus, a branch office across town, or a facility in another city.

After you create a new location, you can add workstations and servers for that location by defining the machine names that are associated with that location.

The topmost location that appears when you launch the JD Edwards EnterpriseOne Deployment Locations Application program (P9654A) is the base location. You cannot change or remove this base location, but you can create or revise locations that are subordinate to it.

When you create a location that is subordinate to another location, the original location is the parent location, and its subordinate location is the child location. For example, if you have a location called Seattle and then create a location called Redmond that is subordinate to Seattle, Seattle is the parent location and Redmond is the child location.

Deployment Groups

You can create a deployment group based on department, team, or function. For example, you might have an administration group, a testing group, a production group, and so on.

Package deployment groups are particularly useful in large enterprises in which scheduling a package for deployment to several individual workstations is very time consuming. In these environments, you can deploy packages much more quickly when you use deployment groups.

A group can contain a subgroup (a group within a group). For example, you might have a group called Quality Assurance that is a subgroup of the larger Development group.

You can help the person who builds and schedules packages by creating easily identifiable names for deployment groups. For example, for a group that includes quality assurance specialists who are responsible for testing, name the group Testing, rather than Green Team.

See Also

- *Setting Up Multitier Deployment*.

Prerequisites

Before you use the JD Edwards EnterpriseOne Deployment Director to deploy a package to individual client workstations, verify that each machine that will receive the package has a record in the Machine Master table (F9650).

Select one of these ways to populate the F9650 table:

- Manually
  For a machine that no user has ever used to sign in to JD Edwards EnterpriseOne, use the JD Edwards EnterpriseOne Deployment Locations application (P9654A) to manually enter a record in the F9650 table.
- Automatically
  The system automatically creates a record in the F9650 table when a user on a new machine signs in to JD Edwards EnterpriseOne for the first time. (The system also automatically updates existing records in the F9650 table each time a user signs in to the workstation.)

The simplest way to populate the F9650 table is to have all users on new machines sign in. In cases in which you need to deploy a package before the users can sign in, you must manually enter machine information. The JD Edwards EnterpriseOne Deployment Locations application enables you to perform this task.

In addition to defining workstations, you can also use the JD Edwards EnterpriseOne Deployment Locations application to enter or revise definitions for these machines:

- Deployment Server
- Enterprise Server
- Data Server
- Java Application Server
- Windows Terminal Server
- Business Services Server

You can enter or revise definitions for these machines in multiple locations, including remote locations.

Forms Used to Define Deployment Parameters

Form Name	**FormID**	**Navigation**	**Usage**
Work with Locations and Machines	W9654AA	Package and Deployment Tools (GH9083), Machine Identification Click Add to add a new location or machine, or click Select to revise an existing location or machine.	Create or revise deployment locations or machines.
Deployment Group Revisions form	W9652AB	Package and Deployment Tools (GH9083), Machine Group Identification Click Add to add a new deployment group, or click Select to revise an existing group.	Create or revise package deployment groups.

Defining Machines

Access the Work with Locations and Machines form.

1. Highlight the type of machine you would like to create. You can choose from:
    - Workstations
    - Deployment Server
    - Enterprise Server
    - Data Server
    - HTML Server
    - Business Services Server
2. Click Add.
3. Enter the appropriate values for the type of machine you are creating.

Workstation

Deployment Server Name

Enter the name of the specific server that is being used for deployment.

When you define a secondary deployment server, options on the Form menu enable you to select path codes, data items, foundation modules, and help items. (These options are not available for the primary deployment server.)

Deployment Server

Primary Deployment Server

Specify whether a deployment server is the primary deployment server for a specific location.

If you have set up a primary deployment server, you cannot access the Primary Deployment Server field when you define a new deployment server. You can change the value in this field only when you revise the primary deployment server definition or when you change the primary deployment server to a secondary server. In this case, you can specify a different server as the primary deployment server.

Server Share Path

Enter the shared directory for this path code. The objects that are stored on a file server will be found in this path.

Enterprise Server

Port Number

Identify the port for a given instance of JD Edwards EnterpriseOne. Because the jde.ini file controls the port to which a workstation will connect, for workstations this port number is for reference only.

Logical Machine Name

Enter the logical machine name that is assigned to this unique machine and port. A machine can be a workstation. Because you can have more than one instance of JD Edwards EnterpriseOne running on a given machine, you must assign a logical machine name that identifies the unique physical machine name and port where this instance runs.

The logical machine name should represent the release and purpose of the machine, such as Financial Data Server-E920 or Distribution Logic Server-E920.

Database Type

Enter the type of database.

Server Map Data Source
Enter the name that identifies the data source.

Installation Path
Enter the path on which JD Edwards EnterpriseOne is installed.

Deployment Server Name
Enter the name of the specific server that is being used for deployment.

Server Availability
This field is visible only in Update mode. Use this field to reset the enterprise server status if a package deployment failed.

Data Server

Data Source Type
Enter the type of database.

HTML Server

Protocol
Specify the method of communication (for example, http).

Server URL
Enter the URL to the web server.

Http Port
Enter the port number of the web server.

Default Login
Enter the login path.

Installation Path
Enter the path on which JD Edwards EnterpriseOne is installed.

Deployment Server Name
Enter the name of the specific server that is being used for deployment.

Business Services Server

The Business Services Server cannot be added through this application. You must use Server Manager to add a new Business Services Server.

See the *JD Edwards EnterpriseOne Tools Server Manager Guide* .

Defining Locations

Access the Location Revisions form by clicking Add on the Work with Locations and Machines form and complete the following fields.

Location
Enter the name of the deployment location.

Location Code
Represent the current location for system deployment.

Parent Location
Enter the name of the parent location.

Defining Package Deployment Groups

Access the Deployment Group Revisions form and complete the following fields.

Deployment Group Name
Enter a profile to use to classify users into groups for system security purposes.

Deployment Group Name
Enter a profile to use to classify users into groups for system security purposes. You use group profiles to give the members of a group access to specific programs.

Some rules for creating a profile for a user class or group are:

- The name of the user class or group must begin with an asterisk (*) so that it does not conflict with any system profiles.
- The User Class/Group field must be blank when you enter a new group profile.

Deployment Group Description
Enter the description for the selected deployment group.

Workstation
Specify the name of the machine on the network.

Workstation Description
Enter a user-defined name or remark.

Deployment Group
Specify a group that is defined to be part of a parent group.

Revising Package Deployment Groups

Access the Deployment Group Revisions form.

Note: When you revise an existing group, you cannot change the group name, but you can change the description.

1. To add to the group, select the last row (the empty one) and enter the name of the workstation or deployment group to which you want to add members.
2. Type the name in the Workstation field or the Deployment Group field, or use the search button for those fields.

   When you use the search button for the Workstation field, the Machine Select form appears. When you use the search button for the Deployment Group field, the Deployment Group Search form appears.

Working with Package Deployment

This section provides an overview of the JD Edwards EnterpriseOne Deployment Director and discusses how to:

- Schedule a package for deployment.
- Revise deployment options.
- Activate a scheduled package.
- Install a scheduled package.

Understanding the Deployment Director

After you define and build a package, use the JD Edwards EnterpriseOne Deployment Director program (P9631// P9631W) to schedule the package for deployment to individual workstations, deployment servers, or enterprise servers. On the specified deployment date, users who are scheduled to receive the package can load the package when they sign in to JD Edwards EnterpriseOne.

Alternatively, you can schedule the package to deployment groups or locations instead of specific machines. Deployment groups are useful in large enterprises that routinely deploy packages to many workstations and servers.

The JD Edwards EnterpriseOne Deployment Director program (P9631/P9631W) simplifies and expedites the process of scheduling and deploying built packages to workstations and servers. The director displays a series of forms that enable you to specify the package that you want to deploy, the deployment destinations, and the deployment time.

After specifying the package that you want to deploy, you specify any of these destinations:

- Client workstation.
- Enterprise Server.
- Deployment Server or Deployment groups.
- Locations.

You can deploy a package either to specific workstations and servers, or you can schedule the deployment based on deployment groups or location. You cannot do both; you must select one of these methods.

You can make the package mandatory, which means that users cannot access the software until they have installed the package. If the package is optional, users will be given the option of installing the package every time that they sign in until they either install or decline the package.

Note: The Mandatory Installation option is applicable to client packages only.

The JD Edwards EnterpriseOne Deployment Director requires that JD Edwards EnterpriseOne already be loaded on the workstation. You can schedule a new full package to replace the existing package, or an update package to be merged with the existing package on the workstation.

The JD Edwards EnterpriseOne Deployment Director uses these tables:

- F9650
- F9651
- F9652

- F9653
- F9654
- F98825
- F988251
- F98826
- F9603
- F96031
- F98826H
- F988259

This table summarizes the function of each form in the JD Edwards EnterpriseOne Deployment Director:

Form Name	Form Usage
Package Deployment Director form	View this form for a description of the JD Edwards EnterpriseOne Deployment Director.
Package Selection form	Use this form to find and select the package that you want to deploy.
Package Deployment Targets form	Use this form to specify the destination for the package. You can select individual client workstations, deployment servers, and enterprise servers, or you can deploy the package to a deployment group or location.
Package Deployment Attributes form	Use this form to enter the date and time that you want to deploy the package. Also specify whether the package is mandatory (that is, it must be installed by every package recipient).
Deployment Client Workstation Selection form	Use this form to select each of the client workstations that will receive the package.
Deployment Server Selection form	Use this form to select each of the deployment servers that will receive the package.
Enterprise Server Selection form	Use this form to select each of the enterprise servers that will receive the package.
Deployment Location Selection form	Use this form to specify the deployment location that will receive the package.
Deployment Groups Selection form	Use this form to specify the deployment groups whose members will receive the package.
Build Selection form	For multitier deployment, use this form to specify the server or client package that you want to deploy to the destination deployment server.
Work with Package Deployment form	Use this form to review and revise the locations and package recipients that you entered through the JD Edwards EnterpriseOne Deployment Director.

Using the Deployment Director

After you have assembled and built the package, defined all machines, and verified the deployment groups, use the JD Edwards EnterpriseOne Deployment Director to specify package recipients and schedule the package for deployment.

Throughout the deployment process, you can select to either proceed to the next form or return to the previous form. Also, regardless of where you are in the process, you can cancel it.

When you schedule a package for deployment to a machine rather than a deployment group or location, you can schedule to deploy the package to client workstations, deployment servers, enterprise servers, or a combination. The forms that appear vary depending on your selection. For example, if you indicate that you want to schedule a package for deployment to client workstations and a deployment server, the forms for selecting specific workstations and deployment servers appear. If you schedule a package for deployment only to client workstations, the server selection form does not appear.

When you access the JD Edwards EnterpriseOne Deployment Director, the Work with Package Deployment form enables you to view deployed package information by either machines, deployment groups, locations, or packages.

Depending on your display selection, the tree displays different information when you expand it. This list describes the information that appears as you expand the tree level by level:

- Machines

  Level One: Client Workstation, Deployment Server, Enterprise Server, and Business Service Application Server headings.

  Level Two: Specific machines under each of these three headings.

  Level Three: Specific packages that are deployed to the machine, if any.
- Deployment Groups

  Level One: Specific groups.

  Level Two: Members of those groups.

  Level Three: Specific packages that are deployed to the group member.

- Locations

  Level One: Specific locations.

  Level Two: Client Workstation, Deployment Server, Enterprise Server, Business Service Application Server, and Remote Locations headings.

  Level Three: Specific machines under the Client Workstation, Deployment Server, and Enterprise Server headings.

  Level Three under Remote Locations only: Defined remote locations.

  Level Four: Specific packages that are deployed to each machine, if any.

  Level Four under Remote Locations only: Client Workstation, Deployment Server, and Enterprise Server headings.

  Level Five under Remote Locations only: Specific machines under the Client Workstation, Deployment Server, and Enterprise Server headings.

  Level Six under Remote Locations only: Specific packages that are deployed to each machine, if any.
- Packages

  Level One: Package names.

  Level Two: Client Workstation, Deployment Server, Enterprise Server, and Business Service Application Server headings.

  Level Three: Package deployment dates and times for each heading.

  Level Four: Specific machines that have deployed that package for that date and time.

Activating Scheduled Packages

After you successfully define a package deployment, you must activate the package so that it is available for installation using the JD Edwards EnterpriseOne Workstation Installation program. If you do not activate the package, it will not be included in the list of available packages when users launch the JD Edwards EnterpriseOne Workstation Installation program.

In some situations, you might need to control which packages are available for installation. If, for example, you have a package that is for the testing group only, you would want to make that package inactive so that it is not available for installation through the JD Edwards EnterpriseOne Workstation Installation program. Instead, you can use JD Edwards EnterpriseOne Deployment Director program (P9631) to schedule this package for deployment to the members of the testing group.

Installing a Scheduled Package

When users receive a package, they can select to install it when they sign in to JD Edwards EnterpriseOne on or after the scheduled deployment date.

If the package is mandatory, users cannot access the system until they load the package.

If the package is optional, users can decline the package or postpone the installation until later. If they decide to postpone the installation, the software launches, and they will be given the opportunity to install the package the next time that they sign in.

Forms Used to Work with Package Deployment

Form Name	FormID	Navigation	Usage
Work with Package Deployment	W9631J	Package and Deployment Tools (GH9083), Package Deployment	Select the package to deploy and review your selections.
Package Selection	W9631C	Package and Deployment Tools (GH9083), Package Deployment. Click Add on the Work with Package Deployment form to launch the Deployment Director. Click Next.	Select the package to deploy.
Package Deployment Targets	W9631B	On Package Selection, click Next.	Select the types of machines on which to deploy the package.
Package Deployment Attributes	W9631D	On Package Deployment Targets, click Next.	Select the type of installation and the time.
Deployment Client Workstation Selection	W9631F	On Package Deployment Targets, select Client Workstation and click Next until the Deployment Client Workstation Selection form appears.	Select the workstations to which the package will be deployed.
Deployment Server Selection	W9631G	On Package Deployment Targets, select Deployment Server and click Next until the Deployment Server Selection form appears.	Select the Deployment Servers to which the package will be deployed.
Build Selection	W9631N	On Package Deployment Targets, click Next until the Build Selection form appears.	Select the server package build to deploy to the destination deployment server.
Enterprise Server Selection	W9631E	On Package Deployment Targets, select Enterprise Server and click Next until the	Select the Enterprise Servers to which the package will be deployed.

Form Name	FormID	Navigation	Usage
		Deployment Server Selection form appears.	
Server Package Deployment Properties Revisions	W9631M	Package and Deployment Tools (GH9083), Package Deployment. Select Machines, and click Find to display information according to machine name. Find and select the deployed package for which you want to modify the options, and then select Properties from the Row menu.	Revise server package deployment options.

Scheduling a Package for Deployment

Access the Package Selection form.

1. Select the package that you want to deploy, and then click Next.
2. On the Package Deployment Targets form, select any of these options to indicate the type of machines to which you want to deploy the package, and then click Next:
   - Client Workstation
   - Deployment Server
   - Enterprise Server
   - Business Services Server

     See *Working with Packages for Business Services (Applications Release 9.2)*.
   - Deployment Group
   - Locations
3. On Package Deployment Attributes, complete these fields:
   - Mandatory Installation
   - Date/Time
4. If you are deploying to workstations, the Deployment Client Workstation Selection form appears. If you are not deploying to workstations, bypass the next step.
5. Find and select the workstations to which you want to deploy the package, and then click Next.

   Select a workstation by double-clicking in its row header. A check mark appears in the row header for each workstation that you select.

   If you are deploying to a deployment server, the Deployment Server Selection form appears. If you are not deploying to a deployment server, bypass the next step.

6. Find and select the deployment server to which you want to deploy the package, and then click Next.

   Select a server by double-clicking in its row header. A check mark appears next to each server that you select.
7. On the Build Selection form, select the server package build that you want to deploy to the destination deployment server, and then click Close.
8. Click Next.

   If you are deploying to an enterprise server, the Enterprise Server Selection form appears. If you are not deploying to an enterprise server, bypass the next step.
9. Find and select the enterprise server to which you want to deploy the package, specify the number of deployment attempts and minutes to wait between retries, and then click Next.

   Select a server by double-clicking in its row header.

   > **Note:** You can deploy an update package only to servers that have the full parent package deployed.
10. On Work with Package Deployment, review your deployment selections.
11. To change any of the selections, click Prev to return to the appropriate previous form.
12. When you are finished reviewing and changing the deployment selections, click End.
13. If you are deploying a server package, find and select the server package on the Work with Package Deployment form, and then select Deploy from the Row menu.

After you schedule the package for deployment, at the specified time on the date that you specified, the package deploys to workstations. This package becomes available to the user when the user signs in.

To schedule a package for deployment to a deployment group or location:

1. On the Package Selection form, select the package that you want to deploy, and then click Next.
2. On the Package Deployment Targets form, select either Deployment Group or Locations, and then click Next.
3. On the Package Deployment Attributes form, complete these fields:
   - Mandatory Installation
   - Date/Time
4. Click Next.

   If you are deploying to a deployment group, the Deployment Groups Selection form appears. If you are deploying to a location, bypass the next step.
5. Find and select the deployment group that you want to receive the package, and then click Next.

   Select a group by double-clicking its row header.
6. If you are deploying to a location, the Deployment Location Selection form appears. Bypass the next step if you are deploying to a deployment group.
7. Find and select the deployment location that you want to receive the package, and then click Next.

   To select a location, double-click the row header.
8. On the Work with Package Deployment form, review the deployment selections.
9. To change any of the selections, click Prev to return to the appropriate previous form.
10. When you are finished reviewing or changing the deployment selections, click End.
11. If you are deploying a server package, find and select the server package on the Work with Package Deployment form, and then select Deploy from the Row menu.

After you schedule the package for deployment, at the specified time on the date that you specified, the package deploys to workstations. This package becomes available to the user when the user signs in.

Scheduling a Package for Deployment for Web (Release 9.2.5)

Starting with Tools Release 9.2.5.0, a web application is available to deploy a package on the web.

1. Access the web application through P9631W or P9621W.
2. Select Deployment from the Row menu. This will take the user to the Work with Package Deployment application.
3. Select + to select a package to deploy on the server or to install it on the client.
4. Select the package and click OK.
5. On the Package Deployment Revision form, select a client machine and the enterprise server.
6. Click OK.

   An Approved status indicates that the package is ready to be deployed to the enterprise server. An Installed status indicates that the package has been deployed previously to the servers.
7. Select Deploy to deploy to one server or Deploy All to deploy to all servers at Approved status.
8. Click on the Client Workstation tab to view the client machines that the package has been pushed to.

Deploying a Package Across Ports (Tools Release 9.2.7.5)

Starting with Tools Release 9.2.7.5, you can deploy a full package to a different port with the same path code. The location of the package must be the same directory on both instances of the different ports. This means that in the `jde.ini` file, under the [BSFN BUILD] BuildArea= <path to build area> points to the same location for both instances with the different ports.

For *IBM i* , the instance for the different ports usually all point to the same package location, `OneWorld\Packages` so deploying to a different port does not need a jde.ini setting change.

For UNIX and Windows, if the instances with the different ports are on the same server, then change the `jde.ini`

[BSFN Build] BuildArea= should point to the location of the package where the package is built for both instances.

Also, ensure that the permission for this directory is set so that the different instance/port can access the `package` directory.

For UNIX and Windows, if the instances with different ports are on different servers, then the system compresses the server package and the deployment package to a different port. The instances on different servers does not apply for Web packages as the web packages do not copy back the compressed files to the Deployment Server. This only works when you build and deploy from the deployment server or the client machine.

Deployment steps to set up the new package in the pathcode location for zero downtime deployment:

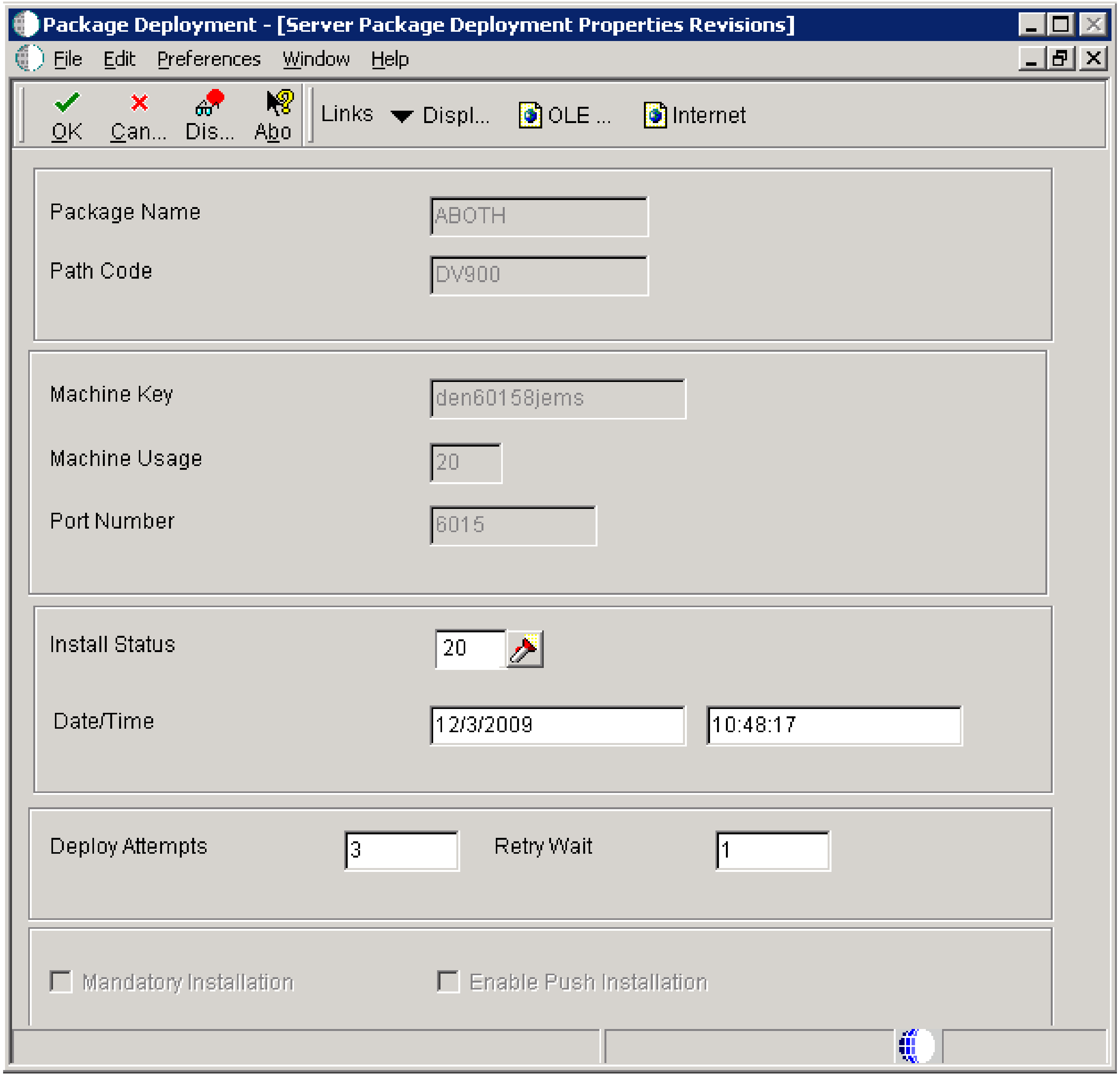
1. Checks if the running EnterpriseOne port is the same as the built package. On the Enterprise Server under `<package name>\ <OS TYPE>` is a `<servername>.inf` that has the port listed. If the port is different than what EnterpriseOne is running, then the system marks it as an across port deploy.
2. (Windows, UNIX) Creates the package name directory under the deploy location `<pathcode>\<bin32\bin64>`.
3. (Windows, UNIX) Copies from the `<package name>\<bin32\bin64>` to the deploy location `<pathcode>\<bin32\bin64>`.
4. (*IBM i*) Checks `jde.ini` if the instance has an addendum.
5. (*IBM i*) Creates a library with `<package name><addendum>`. If there is no addendum, then the system creates the <package name> library.

6. (*IBM i*) Copies from the unique library name of the package to the <package name><addendum> library or if there is no addendum, then the system creates the <package name> library.
7. Creates a package name directory under deploy location `<pathcode>\spec\<packagename>`.
8. Creates the `spec.ini` under the deploy location `<pathcode>\spec\<packagename>`.
9. Changes the current deployed package in `<pathcode>\spec\spec.ini` to the new deployed package.
10. Broadcasts the message that there is a new deployed package.
11. Updates the F96511 table with the package name, port, and enters `33`.

    After the full package is deployed, an update package can be deployed to that full package with the different port.

Revising Deployment Options

Access the Server Package Deployment Properties Revisions form.



Package Name ABOTH

Path Code DV900

Machine Key den60158jems

Machine Usage 20

Port Number 6015

Install Status 20

Date/Time 12/3/2009 10:48:17

Deploy Attempts 3 Retry Wait 1

☐ Mandatory Installation ☐ Enable Push Installation

Package Name
Enter a name for the package.

A package describes the location on the server where components that you want to deploy to workstations or servers reside. Two package types are available:

Full: Contains the full suite of applications (all specifications).

Update: Objects contained in this type of package are loaded after the workstation or server receives the package and the user signs in to the system. If the update package includes just-in-time applications, old versions of the application are deleted from the workstation and replaced by the current version the first time the user accesses that application. Update packages are always deployed on the date and time that are specified by the system administrator.

With the exception of just-in-time applications that are included in an Update package, all packages are a snapshot at a point in time of the central objects for a particular path code. Just-in-time applications are dynamic, not built.

Path Code
Enter the path code.

The path code is a pointer to a set of objects and is used to keep track of sets of objects and their locations.

Deploy Attempts
Enter the number of times to retry the deployment if it fails. This applies to enterprise servers only. If deployment fails on any of the enterprise servers, the application will re-run R98825D.

The default value is 1. Valid values are 1 through 10.

Retry Wait
Enter the number of minutes to wait between retries for a failed deployment attempt. This applies to enterprise servers only. If deployment fails on any of the enterprise servers, the application will wait before re-running R98825D.

Valid values are 1 through 30.

Mandatory Installation
Indicate whether the package is mandatory or optional.

Valid choices are:

Y: The deployment is mandatory. The user must install the package.

N: The deployment is optional to the user.

Enable Push Installation
Select this option to enable the package to be installed through push installation.

Date
Enter a date to deploy updated objects to the listed machine.

Activating the Scheduled Package

Access the Work with Package Deployment form.

1. Click Find.
2. Select from the list the packages that you want to activate or inactivate.

   Alternatively, you can enter the package name in the Package field.
3. Select Active/Inactive from the Row menu.

Installing a Scheduled Package

1. Sign in to JD Edwards EnterpriseOne.

   When you are scheduled to receive a package, Oracle's JD Edwards EnterpriseOne Just-In-Time Installation program launches and the Scheduled Packages form appears.
2. Perform one of these steps:

   - To load the package immediately, bypass to step 3.
   - To decline the package permanently, select Decline from the Row menu.
   - To list all items in the package, select Package Detail from the Row menu.
   - To load the package at another time, click Close. If the package is mandatory, you will be unable to access the system until you load the package.
3. To load the package, select one or more packages that you want to install and click Select.

   The JD Edwards EnterpriseOne Workstation Installation program loads the package. If you selected more than one package, the program installs them sequentially. When the installation is complete, the software launches.

Deploying a Server Package

This section provides overviews of server package deployment and deployment to web servers, lists prerequisites, and discusses how to:

- Deploy a server package.
- Monitor package deployment.

Understanding Server Package Deployment

The process for deploying a server package is nearly identical to that for deploying a package to a workstation. In both cases, you need to assemble, define, build, and schedule the package for deployment by using the JD Edwards EnterpriseOne Package Assembly (P9601/P9601W), Package Build Director (P9621/P9601W), and Deployment Director (P9631) programs.

After you schedule a server package for deployment, to deploy to the servers, you must complete an additional step to launch the batch program that you use on the client or deployment server. For web packages, you must use the Deploy button at the bottom of the page. You must perform these tasks whenever you deploy a package to an Enterprise Server or Deployment Server.

Note: Starting with Tools Release 9.2.6.0, when you deploy a full server package, no locks are placed on the processes that are running. Any ongoing UBE processing will continue to process using the old package specs. When a new UBE is submitted, it will use the current package that is deployed. Also, web users will continue to be logged in to their session using the old package. If they sign off and sign back in, they will switch to the current package. You can deploy update packages that contain UBE-only, APP-only, or UBE/APP-only packages at any time. Their deployment does not affect any UBEs that are currently running or those that are submitted. All other types of server packages should be deployed only when necessary, because the enterprise server is not available to process business applications and batch processes during the installation process. The enterprise server does not actually shut down during package installation of the update package. Instead, the system queues any UBE jobs that are submitted to the enterprise server that are in the package and runs them as soon as the installation finishes.

Note: Starting with Release 9.2.7.0, after the package is deployed, the process will submit the Table Conversion UBE R98405W that runs on the server. This UBE runs the table conversions for full and update package. For a full package build, the TC UBE converts all tables listed in the F98405 table. For update packages, table conversion runs only if objects in the package are tables.

To further minimize impact on the network and users, if the development environment is on the same enterprise server as the production environment, consider preventing developers from moving their own objects through server packages. Instead, require that an administrator perform this function.

To deploy a server package using the applications on the client workstation or deployment server, select Deploy from the Row menu on the Work with Package Deployment form.

When you select Deploy from the Row menu, the system determines the batch programs to call based on what is currently selected on the Work with Package Deployment form.

- If a specific Enterprise Server is selected, the system launches the Enterprise Server Deployment batch program (R98825D).
- If the Enterprise Server folder is selected, the system launches the Enterprise Server Deployment batch program for every Enterprise Server that has a package scheduled for deployment..
- If a specific deployment server is selected, the system launches the Multi Tier Deployment batch program (R98825C).
- If the deployment server folder is selected, the system launches the Multi Tier Deployment batch program for every deployment server that has a package scheduled.
- If a specific package is selected, the system launches the Multi Tier Deployment batch program, and then the Enterprise Server Deployment batch program for the selected package.
- If you sort by packages and the Deployment folder is selected, the system launches both the Multi Tier Deployment batch program and Enterprise Server Deployment batch programs for all packages.
- If a specific workstation or the Workstations folder is selected, the Deploy option is unavailable.

When the system launches a batch program for all servers or all packages, deployment does not occur unless the package has been previously scheduled for a specific server. A full package can be deployed to all servers. However, an update package can be deployed only to servers that already have the parent package deployed. Also, update packages cannot be deployed if the parent package is an inactive or not-deployed full package.

When the system launches the batch program to deploy an update package that is UBE-only, APP-only, or UBE/APP-only package to an enterprise server, the batch process:

1. Verifies that the enterprise server deployment location is the same as the Windows client submitting the package.
2. Changes the enterprise server status to **Pre Deploy.**

This is done by changing the `MDMCHRCDNM` column to **50** in the F9651 table.

3. If this is an APP-only or UBE/APP-only package, then it waits one minute for the HTML server to find the deployment record.
4. Sends lock messages to the metadata kernel for the UBEs in the package on each selected enterprise server.
5. Once the package is being deployed, the UBEs in the package cannot be submitted.

   This is done by changing the `MDMCHRCDNM` column to **10** in the F9651 table.
6. Updates the specifications in the database.
7. Sends unlock messages to all the enterprise servers to unlock the UBEs that were in the package.
8. Marks the servers as available.

   This is done by changing the `MDMCHRCDNM` column to **30** in the F9651 table.
9. Updates the F96511 table with the new package and spec data source information.

   This information is used by the web servers.

Note: A deployed package can be deployed multiple times to the same or different servers.

When the system launches the batch program to deploy all update package with Business functions, NER and tables, and other type objects to an enterprise server, the batch process:

1. Verifies that the enterprise server deployment location is the same as the Microsoft Windows client submitting the package.
2. Changes the enterprise server status to **Pre Deploy.**

   This is done by changing the `MDMCHRCDNM` column to **50** in the F9651 table.
3. Waits for five minutes.
4. Sends lock messages to all the selected enterprise servers.
5. Once the servers are locked, the batch process marks them as unavailable.

   This is done by changing the `MDMCHRCDNM` column to **10** in the F9651 table.
6. Copies the BSFN executables from the package location to the live path code location.
7. Sets the spec.ini file to the new package and spec data source.
8. Sends unlock messages to all the enterprise servers.
9. Marks the servers as available.

   This is done by changing the `MDMCHRCDNM` column to **30** in the F9651 table.
10. Updates the F96511 table with the new package and spec data source information.

    This information is used by the web servers.
11. (Release 9.2.7.0) Checks if there are tables in the package.
12. If there are tables in the package, the system calls B98405WS which then submits R98405W UBE to run on the server.
13. The R98405W UBE checks if any tables are in update package which is listed in the F98405 table for Table Conversions.
14. If tables are listed in the F98405 table, then the process runs the table conversion for each table.

Note: A deployed package can be deployed multiple times to the same or different servers.

Starting with Tools Release 9.2.6.0, when the system launches the batch program to deploy a full package to the Enterprise Server, the process:

1. Sets the F9651 field to `mdmchrcdnm= 50` to prepare the server for deployment..

2. Opens the <pathcode>/spec.ini file and changes the <pathcode>_Package setting to the new package name: **DV920_Package=DV920FA**.
3. Sends a broadcast message that a package change was performed.
4. Sets the F9651 field to `mdmchrcdnm= 30` to indicate that deployment is completed.
5. Updates the record in the F96511 table with the Enterprise Server information, port number, pathcode, package name, and type 30 to indicate that this is the deployed package.
6. (Starting with Tools Release 9.2.5.0) If the package is also a client package, insert a record in the F98825 table with these details:

   ```
   MKEY=”CLIENT" UPJDEPKGNM=<package name> UPPATHCOD=<pathcode> UPINPKST=’20’ Date and time
   ```
7. For the previously deployed package with the details MKEY= CLIENT and UPINPKST=20, update the status to UPINPKST=30.
8. (Release 9.2.7.0) Checks if it is a full package build. If yes, the system submits TC UBE R98405W to run on the server.
9. The R98405W UBE checks if any tables are listed in the F98405 table to convert.
10. If tables are listed in the F98405 table, the process initiates the table conversion on each table.

Note: A deployed package can be deployed multiple times to the same or different servers.

A server package with the specs built in a shared mode can be deployed to a web server. This process of deploying to the web server is automatic and does not require any end user intervention. The web servers pool the package information from the JD Edwards EnterpriseOne logic server. It compares the package manifest from the spec tables to the one in its serialized database and makes the necessary updates.

(Prior to Tools Release 9.2.6.0) During server package deployment, the business function (BSFN) DLL's, SRVPGMs, .so objects, or .sl objects of the live package are replaced by the objects from the built package. However, if a deployment fails, you may have a mismatched set of BSFNs and specs. With 8.96 JD Edwards EnterpriseOne clients, you can back up the existing BSFN objects. If the deployment fails, you can restore the BSFN objects. The option to back up the live BSFN objects before deployment can be enabled through the Build Settings within Server Manager.

(Prior to Tools Release 9.2.6.0) For *IBM i* , the BSFN objects in PY920 are copied into the $PY920 library. For Microsoft Windows and UNIX, the BSFN and spec objects in PY920 are copied to the PY920_BACK folder. Clients can restore BSFN objects by copying the objects from the backup location to the live folder. They can restore the specs by changing the package name to the previous package in the spec.ini file.

(Starting with Tools Release 9.2.6.0) During the package build, the deploy folders are set up under <pathcode>\bin64\<packagename and <pathcode>\spec\<packagename> and are populated with the files needed. The deployment of the package, only changes the <pathcode>\spec\ spec.ini to the current package and no other process is carried out.

Understanding Deployment to Web Servers

Web servers have the ability to determine which package is to be deployed on the default enterprise server and generate serialized objects on demand. The web servers also have the capability to compare the contents of a package with that of a new package and make the necessary adjustments, such as deleting obsolete serialized objects. This is done by using a package manifest.

A package manifest is a spec record that is created by the package build process. The manifest describes the package and its contents. The serialized object generator compares the manifest from the deployed package on the enterprise server to one that is created during the generation process and makes the appropriate changes.

(Prior to Tools Release 9.2.6.0) The process for deploying packages to web servers is:

1. The web servers check the F9651 and F96511 tables every minute for any change in the package.
2. The one minute interval is changed to five seconds once the enterprise server is in a **Pre Deploy** state.
3. All JD Edwards EnterpriseOne users are prevented from running any applications once the enterprise server is in a **Locked** state.
4. The web server compares the package manifest in the F98770 on the enterprise server with the one in its serialized object database after the package is deployed and the enterprise server is in an **Available** state.
5. The web server synchronizes the serialized object database with the deployed package.
   The contents of the serialized object database are deleted for a new full package deployment. Only the objects in the update package are deleted for an update package deployment.

Note: Deploying server packages to web servers is supported only in shared spec server packages. You can have only one path code and one package per Java node. Serialized objects should not be shared with nodes running dissimilar packages and path codes. Starting with Tools Release 9.2.6.0, package deployment no longer affects the Web Server. The users will continue to use the package they signed in with. If they sign off and sign in back in, they will be moved to the new package that was deployed.

Prerequisites

Before you complete the tasks in this section:

- Assemble the server package.
- Define the server package.
- Build the server package.
- Schedule the package for deployment to the appropriate server.
  These tasks can be done from the Web Server P9631W or the deployment server/client workstation P9631 applicaton.

Forms Used to Deploy Server Packages

Form Name	FormID	Navigation	Usage
Work with Package Deployment	W9631J	Package and Deployment Tools (GH9083), Package Deployment	Select the package to deploy and review your selections.
Monitor Deployment	W9632A	Package and Deployment Tools (GH9083), Deployment Monitoring. Click Find or enter the Package Name or Path Code and click Find to display package	Monitor the status of a package deployment while it is running or retrieve the R98825D pdf and deployment logs after completion.

Form Name	FormID	Navigation	Usage
		deployment records. Find and select the deployed package that you want to review, and then select either Display PDF or Display Logs from the Row menu.	

Deploying a Server Package

Access the Work with Package Deployment form GH9083/Package Deployment, from the workstation.

1. Locate the server package that you want to deploy.

   Alternatively, select the enterprise server or, if the package is scheduled to deploy to more than one server, the Enterprise Server folder.
2. Select Deploy from the Row menu.
3. On Report Output Destination, select On Screen.
4. Click OK.
5. (Release 9.2.7.0) When package is deployed, the process submits the R98405W program on the server to process table conversion.
6. When the table conversions are complete, the deploy status will change to Deployed/TC Completed. Refer to *Document 2554489.1* for more information on table conversions.

(Starting with Tools Release 9.2.6.0) Access the Work with Package Deployment form, GH9083/Package Deployment, from the Web Server:

1. Locate the server package that you want to deploy.
2. Select a + and select your package.Click Exit.
3. Select Enterprise Server and the servers you from which you would like to deploy the package.

   The servers are listed at the bottom of the page.
4. To deploy the package to a single server. select one server and click Deploy. To deploy the package to all the servers, select Deploy All.
5. (Release 9.2.7.0) The deployment status changes to Processing to indicate that the button was clicked.
6. When the deployment is complete, the deploy status changes to Deployed/TC submitted. This indicates that the deployment was successful and the table conversion was submitted.
7. When the table conversion is complete, the deploy status changes to Deployed/TC Completed.
8. Click the Refresh button to update the status.

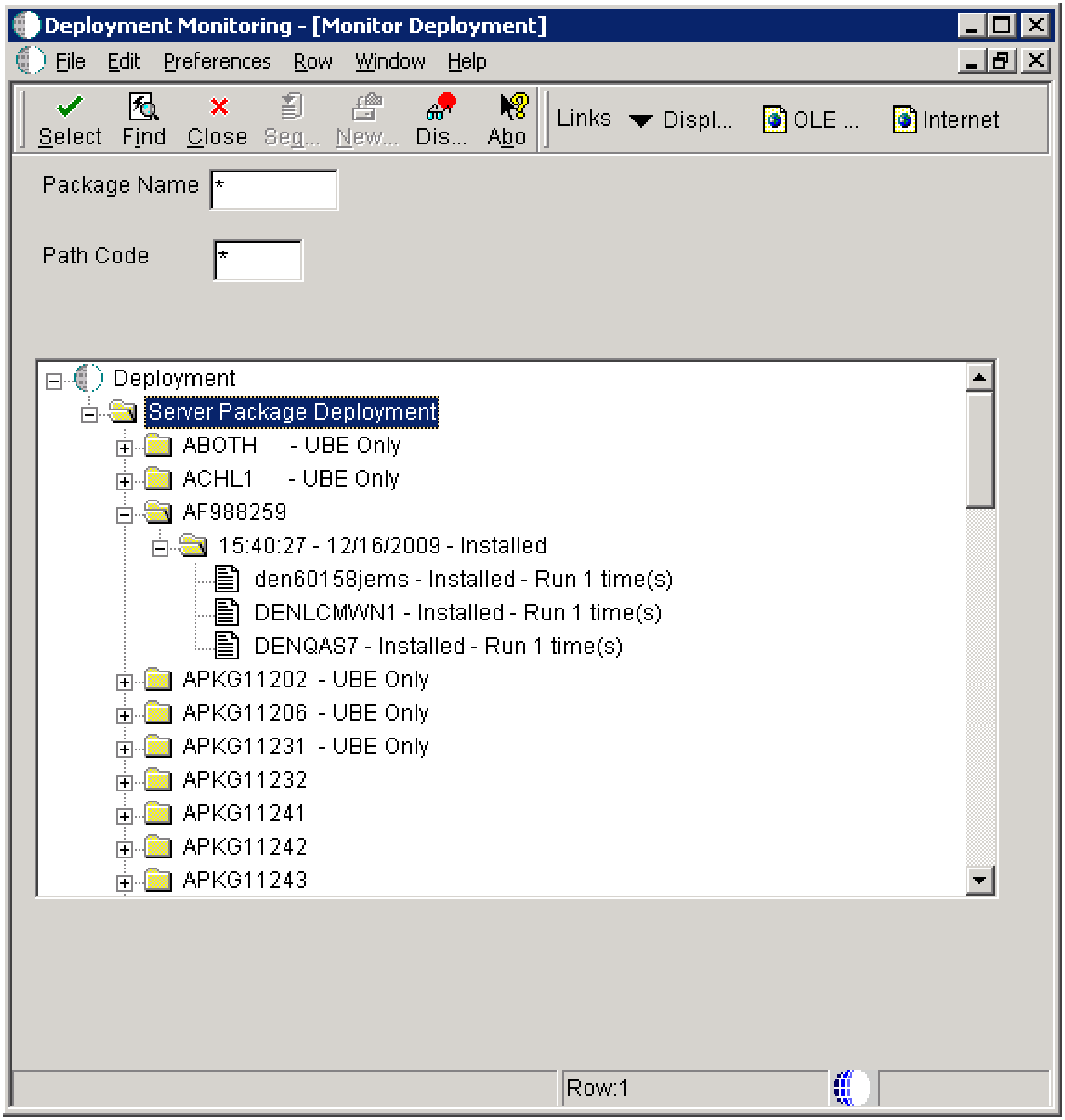
   Note: Refer to *Document 2554489.1* for more information on table conversions.

Monitoring Package Deployment

While the server deployment (R98825D) is running, you can use the Monitor Deployment application (P9632) to see the current status of the deployment process. For example, the Monitor Deployment application will show if the system is waiting for locks, how many times the deployment (R98825D) has been run if using the Retry option, and which enterprise server deployments failed. You can also retrieve the R98825D pdf and the deployment logs, both local and from the servers, by using a row exit within this application.

Note: This is applicable only for the Deployment server/ Client Workstation.

Access the Monitor Package Deployment form.


Deployment Monitoring - [Monitor Deployment]
File Edit Preferences Row Window Help
Select Find Close Seq... New... Dis... Abo
Links Displ... OLE ... Internet

Package Name *

Path Code *

```
Deployment
  Server Package Deployment
    ABOTH     - UBE Only
    ACHL1     - UBE Only
    AF988259
      15:40:27 - 12/16/2009 - Installed
        den60158jems - Installed - Run 1 time(s)
        DENLCMWN1 - Installed - Run 1 time(s)
        DENQAS7 - Installed - Run 1 time(s)
    APKG11202  - UBE Only
    APKG11206  - UBE Only
    APKG11231  - UBE Only
    APKG11232
    APKG11241
    APKG11242
    APKG11243
```

Row:1

1. Click Find.

2. Open the tree structure to view individual deployment records.

   If the server package deployment is still processing, a status of **Processing** or **Waiting for Locks** will appear next to the server. If the server package deployment failed, the deployment error will appear next to the server that failed.
3. Select the package deployment record that you want to monitor.

   Alternatively, you can enter the package name in the Package Name field.
4. Select Display PDF from the Row menu to view the deployment report.
5. Select Display Logs from the Row menu to view the build and deploy logs.

   You can view the ClientPkgBld.log and the SvrPkgBuild.log from the deployment server, and the SvrPkgBuild.log from the enterprise server. The log from the enterprise server will include the server name at the beginning of its filename. For example, den60158jems_SvrPkgBuild.log

Installing Workstations from CD

This section provides an overview of how to install workstations from CD, lists a prerequisite, and discusses how to:

- Define the CD Writer location.
- Deploy a package to the CD Writer location.
- Create the installation CD.

Understanding How to Install Workstations from CD

If your system includes a CD writer, you can build and deploy packages to the CD writer location. After copying the package to a CD, you can then use the CD as a portable deployment tier from which to perform workstation installations. That is, you can run from the CD the setup.exe program that launches the JD Edwards EnterpriseOne Workstation Installation program.

You can set up your enterprise so that you can deploy packages to the CD writer and install the software from a CD.

The first step in the process of configuring your system for deployment from CD is to define the CD writer location if it is not already defined. In this step, you essentially create a pseudo deployment server from which you will later copy package data onto the CD by using the software for your CD writer.

When you define the CD writer location in the Machine Identification application, you must also add the correct path codes to the Environments exit.

The process for defining this location is identical to the process for defining any other new deployment server.

Prerequisite

Assemble, define, and build the package that you want to write to the CD.

Forms Used to Install Workstations from CD

Form Name	FormID	Navigation	Usage
Deployment Server Revisions	W9654AC	Package and Deployment Tools (GH9083), Machine Identification Subordinate to the appropriate location, select Deployment Server. Click Add to add a new machine.	
Work with Package Deployment	W9631J	Package and Deployment Tools (GH9083), Package Deployment	Select the package to deploy and review your selections.

Defining the CD Writer Location

Access the Deployment Server Revisions form.

1. Enter the name of the machine on the network in the Machine Name field.
2. Enter the release number as defined in the Release Master in the Release field.
3. Enter the primary user for the listed machine in the Primary User field.
4. Enter the shared directory for the path code in the Server Share Path field.
5. If you want to specify a location for data, a foundation, or help files, do so by choosing Data, Foundation, or Helps from the Form menu.

   If you do not specify a location for data, foundation, or helps, the system uses the default locations.
6. Click OK.
7. Click Close to quit the Work with Locations and Machines form.
8. In Windows Explorer, locate the folder named Client Install.
9. Copy this folder by dragging the folder to the CD writer location.

   The location is the server share path that you entered on the Deployment Server Revisions form.

Deploying a Package to the CD Writer Location

After you define the CD writer as a deployment server, you are ready to deploy a package to the CD writer location that you specified. This task involves these two procedures:

- Deploy to the CD writer location the package that you want to write to the CD.
- Modify the Install.inf and Package.inf files in preparation for writing the package contents to the CD.

Access the Work with Package Deployment form.

1. Click Add.
2. Complete the forms in the JD Edwards EnterpriseOne Deployment Director in the same way as you would for any other package.
3. From the Work with Package Deployment form, find and select the package that you just scheduled for deployment, and then select Deploy from the Row menu to deploy the package.

After you deploy the package to the CD writer location, the directory structure for that location will look similar to this example:

```
Multitier\package_inf\Appl_B.inf

Multitier\systemcomp\system.cab

Multitier\datacomp\data.cab

Multitier\helpscomp\helps.cab

Multitier\Appl_pgf\Package\Appl_B

Multitier\package_inf\Appl_B.inf
```

In the previous example, Multitier is the name of the server share path. Appl_B is the package name.

Note: The server share path name is not included when you copy folders to the CD. In the previous example, the items that are copied to the CD are \package_inf\Appl_B.inf, \systemcomp\system.cab, and so on.

To modify the Install.inf and Package.inf files:

1. In Windows Explorer, find the CD writer location and open the folder that contains the package that you deployed.
   This folder has the name that you entered in the Server Share Path field on the Deployment Server Revisions form when you defined the CD writer location. In the previous example, the server share path name is Multitier.
2. Open the Client Installation folder, and then open the file Install.inf.
   That is, double-click the icon for the file to launch the Microsoft Notepad application.
3. In the section [FileLocations], modify the line so that two periods and a backslash (\) precede the package_inf entry.
   The line should look like this example after you modify it:
   ```
   [FileLocations]

   PackageInfs=..\package_inf
   ```
4. Similarly, open the Package_inf folder and open the package name.inf file.
   In the previous example, this file is named Appl_b.inf.
5. In the section [SrcDirs], modify each of the lines so that two periods and a backslash (\) precede each entry.
   After modification, the [SrcDirs] section should look similar to this example:
   ```
   [SrcDirs]

   SAPPL_PGF=..\APPL_PGF\package\APPL_B

   SSYS=..\systemcomp

   SAPPL_PGFDATA=..\datacomp

   SHELP=..\helpscomp
   ```

Creating the Installation CD

After you deploy the package to the CD writer location and modify the Install.inf and Package.inf files, you are ready to copy the package contents to the CD. Use the software that came with your CD writer to accomplish this process, which typically involves copying the package contents to the CD. Refer to the documentation that came with your CD writer for more information about this process.

You copy the package to the CD by copying the subdirectories that are subordinate to the server share path directory. The server share path directory is not created on the CD. (In the previous example, the server share path directory is called Multitier, and it is the same name that you entered in the Server Share Path field on the Deployment Server Revisions form.

When you are finished copying the directories to the CD, the CD should contain these directories:

- Appl_pgf (contains package information).
- datacomp (contains the database cabinet file).
- helpscomp (contains the helps cabinet file).
- systemcomp (contains the foundation cabinet file).
- package_inf (contains the package.inf file).
- Client Install (contains the JD Edwards EnterpriseOne Workstation Installation program).

Note: Actual names might not be the same as those listed because each system might be different.

8 Working with Packages for Business Services (Applications Release 9.2)

Understanding Packages for Business Services

After business services are created, they need to be built and deployed for consumption. The package build process builds and creates the necessary files for development client, WebLogic Server (WLS), and Websphere Application Server (WAS) consumption. The client installation process deploys business services to all development clients. The package deployment process deploys the business services archive (.ear) files to the preconfigured J2EE containers on WLS and WAS.

JD Edwards EnterpriseOne supports building JAX-WS based business service packages.

JAX-WS-Based Business Service Packages

Java API for XML-based web services (JAX-WS) is an API for building standards-based web services and clients that are message oriented. JAX-WS supports transmission of SOAP messages over HTTP. JAX-WS delegates data binding-related tasks to the Java API for XML Binding (JAXB). JAXB provides support for Java to XML mapping, additional support for less used XML schema constructs, and also provides bidirectional customization of Java and XML data binding.

JD Edwards EnterpriseOne supports building the following combinations of JAX-WS based business services packages:

- JAX-WS package for WLS with JDeveloper 11g.
- JAX-WS package for WAS with JDeveloper 11g.
- JAX-WS package for WLS with JDeveloper 12c.
- JAX-WS package for WAS with JDeveloper 12c.

The business services package includes the JAX-WS based WSDL artifacts for the published business services and the newly developed JAX-WS business service consumer proxies.

Note: You can build JAX-WS business services packages for both WLS and WAS at the same time by selecting both the WLS and WAS options in the Business Services Assembly Application.

Note: Beginning with Tools Release 9.2.1.0, IBM WebSphere Application Server 9.0 is supported.

Note:

- *Assembling Business Services for Package Build* in this document for detailed steps.
- *"Configuring Business Services Server Security for JAX-WS Based Business Services" in the JD Edwards EnterpriseOne Tools Business Services Server Reference Guide* for more information about configuration and security of the Business Services Server.

Assembling JD Edwards EnterpriseOne Business Services

This section lists prerequisites and discusses how to assemble business services for package build.

Prerequisites

Before you complete the tasks in this section:

- Use Server Manager to create J2EE business service container(s) for the Business Services Server.
- Use Server Manager to set up Server Manager users.

  In order to deploy the package successfully, the JD Edwards EnterpriseOne user must be a valid Server Manager user. The user cannot deploy the package if the JD Edwards EnterpriseOne user's credentials are not valid for Server Manager.

  See the *JD Edwards EnterpriseOne Tools Server Manager Guide* .
- If you have multiple security servers, you must set up JD Edwards EnterpriseOne Trusted Nodes for a successful deployment of business services.

  See *"Setting Up a Trusted Node Configuration" in the JD Edwards EnterpriseOne Tools Security Administration Guide* .
- If you are building a business services package for WebLogic Server with JDeveloper 11g, JDeveloper 11.1.1.5 needs to be installed on the JD Edwards EnterpriseOne client that you are using to build the business services package.
- In the jde.ini, set the JDeveloperVersion:
  - To **11.1.1** if you are assembling a business services package for WLS 11g.
  - To **12.1.2** if you are assembling a business services package for WLS 12c.

These additional prerequisites are required if you are building a JAX-WS based business services package for WLS, WAS, or both WLS and WAS:

- Ensure that you have JDeveloper 11.1.1.5 installed on the JD Edwards EnterpriseOne client or deployment server machine that you are using to build the business services package, and set JDeveloperVersion to 11.1.1 in the jde.ini file.
- Ensure that you have JDeveloper 12.1.2 installed on the JD Edwards EnterpriseOne client or deployment server machine that you are using to build the business services package, and set JDeveloperVersion to 12.1.2 in the jde.ini file.

Assembling Business Services for Package Build

Enter P9603 in the Fast Path.

1. On the Assemble Business Services form, do one of the following:

- Select the WLS option and provide the JDeveloper 11.1.1.5 install path to build for WLS 11g.
- Select the WLS option and provide the JDeveloper 12c install path to build for WLS 12c.
- Select the WAS option and provide the JDeveloper 11.1.1.5 or JDeveloper 12c install path to build for WAS.

  > **Note:** The package for WAS built with JDeveloper 12c is supported only on a WAS 8.5 container hosted with JDK 1.7, or beginning with Tools Release 9.2.1.0, it is also supported on a WAS 9 container hosted with JDK 1.8.

- Select both WLS and WAS options and provide the JDeveloper 11.1.1.5 install path to build for WLS 11g, and WAS 7 or 8.5.
- Select both WLS and WAS options and provide the JDeveloper 12c install path to build for both WLS 12c and WAS 8.5 with JDK 1.7 or (Tools Release 9.2.1.0) WAS 9 with JDK 1.8.

2. In the Path Code field, enter the path code of the package that you plan to build and tab to the next field.

   > **Note:** At this point, the application retrieves all the available business services from F98602 and F98603. If you have used this application before, the application also retrieves the values for the JDeveloper Install Path field from the JDE.INI file.

3. If this is your first time in the application, you must manually enter the JDeveloper install path (on the package build machine) in the JDeveloper Install Path field.

   The JDeveloper 11g install path should include the root folder of the JDeveloper 11g installation (for example, C:\Oracle\Middleware), and not the JDeveloper folder.

   The JDeveloper 12c install path should include the root folder of the JDeveloper 12c installation (for example, C:\Oracle\Middleware\Oracle_Home), and not the JDeveloper folder.
4. When you enter the install path, P9603 verifies the actual location and version.

   If this path is correct, P9603 adds this information to the jde.ini:

   ```
   [MTR VALIDATION]
   JDeveloperInstallPath=<Install path specified by P9603>
   ```

   > **Note:** If the path or version is incorrect, an error appears; the Close button is disabled until you enter the correct path.

5. In the grid, select the business services that you want to expose as a web service and click the Select button.

   You can also double-click the row headings of the business services that you want to expose.

   A check mark appears by each business service that is selected.
6. Click Select again or double-click the row header to hide the web service.
7. Click Close to close the application.

Assembling a Package that Contains Published Business Services

This section discusses how to assemble a business service package.

Assembling a Business Service Package

To set the processing options for Package Assembly, go to the Package and Deployment Tools menu, right-click the Package Assembly application (P9601), and select prompt for values.

1. Set the processing option entitled Business Services to **1.** This processing option is blank by default.
2. Select OK.
3. On the Work with Packages form, begin the assembly process.

   See *Assembling Packages*.

Building a Package with Published Business Services

This section provides an overview of the build process, prerequisites, and discusses how to:

- Define a package build with published business services.
- Resubmit the package build.

Understanding the Build Process for a Business Service Package

When building a business services package for WLS & WAS, the JD Edwards EnterpriseOne System:

1. Creates the \\work\sbf\sbfbuild.ini, which defines the paths to the exposed methods.
2. Creates the Ant scripts, logtimestamp.xml and build.xml, in the \\work\sbf directory.
3. Runs the build.xml Ant script to extract source.

   When the extract occurs, Unjar_BusinessService.log is generated in the \\work\sbf directory.
4. Creates Java Web Service (JWS) files.

   The JWS files are created on the EnterpriseOne Deployment Server in the E1_Install_Path\Pathcode \Package\PackageName\java\sbf\sbfjaxwswls directory for WLS and at E1_Install_Path\Pathcode\Package \PackageName\java\sbf\sbfjaxwswas directory for WAS.

   A JWS file is created for each published business service, and it has annotations that describe the published business service. The selected methods are listed in the created published business service. The JWS files are required to create the WSDL artifacts for the published business services, and they should not be edited manually.
5. Creates the Web Service Inspection Language (WSIL) file, which is used for Business Process Execution Language (BPEL).
6. Creates Ant scripts for WLS and WAS.

   These scripts are named build.compile.xml, build.xml, and sbfwebservices.xml. They are created within the \ \work\sbf\wls directory for WLS and within the \\work\sbf\was directory for WAS.
7. Runs the build.xml Ant script.

   The build.xml Ant script:

   a. Creates javadoc.
   b. Compiles the business services java source files.
   c. Invokes the sbfwebservices.xml Ant script to run the wsgen Ant task

Invokes the sbfwebservices.xml Ant script to run the wsgen Ant task to create the web services artifacts (such as the WSDL file).

d. Generates the required deployment descriptors, and assembles the business services classes into an .ear file that is compatible with WLS and WAS.

When the build process finishes, an E1Services-Pathcode-wls.ear file is created for WLS, and an E1Services-Pathcode-was.ear file is created for WAS on the EnterpriseOne Deployment Server within the E1_Install_Path\Pathcode\Package \PackageName directory.

Note: For WLS, review the wls_BusinessService.log in the \\work\sbf\wls directory and for WAS, review the was_BusinessService.log in the \\work\sbf\wls directory to verify that the build was successful. If the build is successful, *Build Successful* appears at the bottom of the log. If the build failed, *Build Failed* appears at the bottom of the log.

Prerequisites

Before building the package, verify that logging is turned off. When the jdeproperties.log file is set for logging in the \system\classes folder and a package build is submitted, the build process is slowed down. It is not recommended to have logging turned on during package builds.

Note: When developing business services using JDeveloper 12c, you must select the Tools-> Initialize EOne Workspace option after JDeveloper is launched from OMW.

Defining a Package Build with Published Business Services

To define a package build with published business services:

1. Enter P9621 in the Fast Path or go to the Package and Deployment Tools menu and select Package Build.
2. Use the Package Build application to define build properties for the package.
3. On the Package Build Location form, select Client as the Build Location.

   Note: Business services are not built for server-only packages.

The JVM's virtual memory is set to a maximum of 512 MB. You can change this using the following jde.ini settings:

```
[PACKAGE BUILD]

JavacMaxMemorySize= 512 MB

JavadocMaxMemorySize= 512 MB
```

Note: The javadoc task with JDeveloper 12c (JDK 1.7) requires more memory to load the business services and to create javadoc during the business service build process. The following error can occur if you use the default memory of 512 MB:

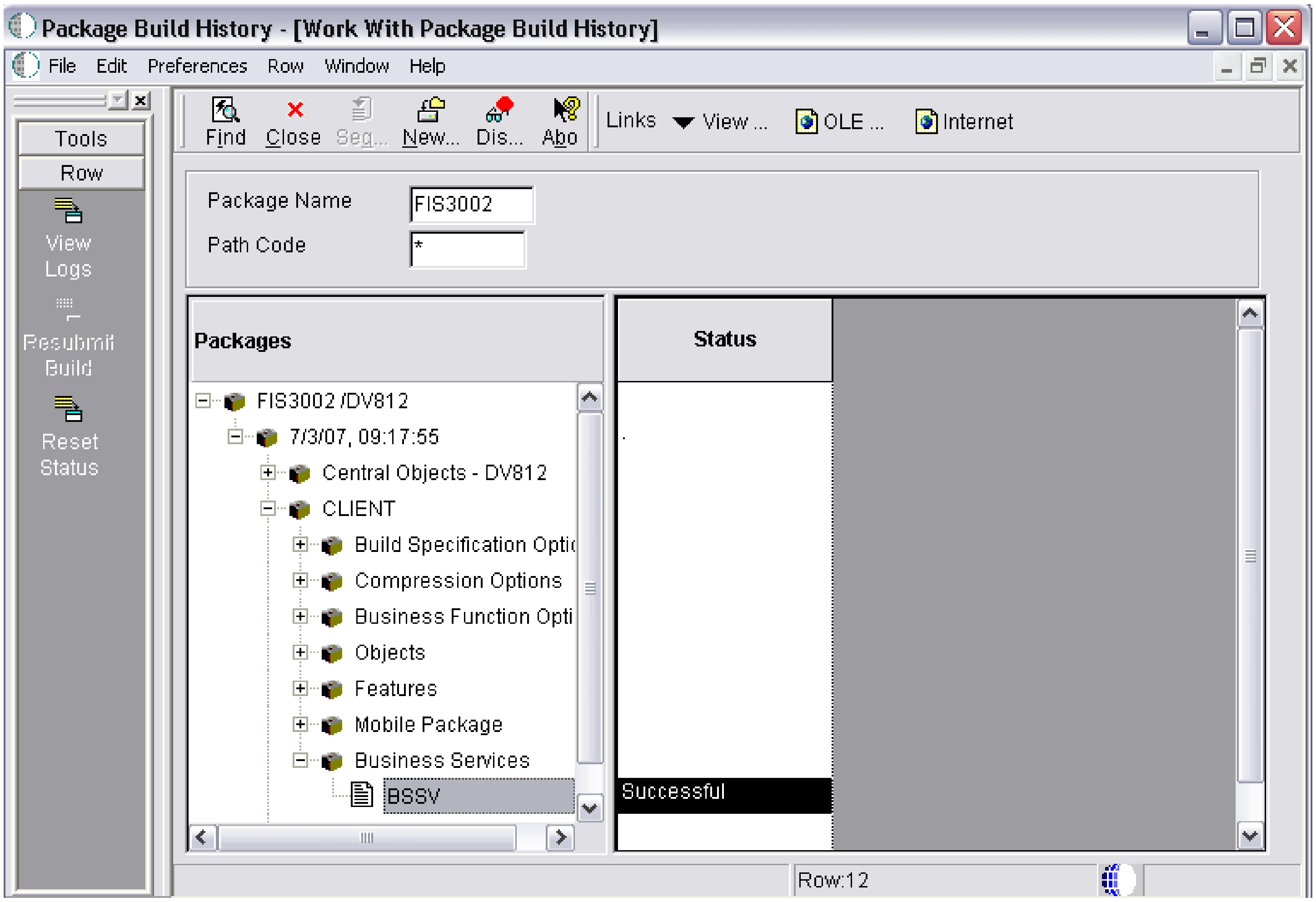
```
javadoc: error - java.lang.OutOfMemoryError: Please increase memory.
```

When building business services packages with JDeveloper 12c, increase the JDE.INI entry **JavadocMaxMemorySize** to 760 MB.

Resubmitting the Package Build

If errors occur during the package build, you will need to reset the status and resubmit the package build.

Enter P9622 in the Fast Path or go to the Package and Deployment Tools menu and select Package Build History.



Package Name FIS3002
Path Code *

FIS3002 /DV812
7/3/07, 09:17:55
Central Objects - DV812
CLIENT
Build Specification Optic
Compression Options
Business Function Opti
Objects
Features
Mobile Package
Business Services
BSSV

Successful

1. On the Work with Package Build History form, enter your package in the Package Name field and select Find.
2. In the tree structure, expand your package name and CLIENT.
3. Click Business Services and select your business service.
4. Select Reset Status from the Row menu to reset the status of your business services.
5. Select Resubmit Build from the Row menu.

Deploying the Package to the Business Services Server

This section provides overviews of the deployment process for WAS and WLS, and discusses how to deploy the business services.

Understanding the Deployment Process for WAS

This is an overview of how business services are deployed to the Business Services Server for WAS.

1. When you click Deploy, the R98825F runs.

   > **Note:** If you are deploying the package to both the enterprise server and the business services server, you select the enterprise server and click Deploy. R98825D deploys the package to the enterprise server and then calls R98825F to deploy the package to the business services server.

2. The R98825F creates the scfjar folder.
3. The scf_manifest.xml is created in the scfjar folder.

   This xml contains information that the package deployment process uses to communicate with the Server Manager.
4. The WAS .ear files are copied to the scfjar folder.
5. The contents of the scfjar folder are combined to form the bssv_timestamp.jar file.

   > **Note:** If errors occur, they are logged to the jas.log or jasdebug.log files that are configured through the jdelog.properties file in the E1_Install_Path\system\classes path. Errors are also logged in the Server Manager EnterpriseOne Agent logs on the machine on which WAS is installed and under which the business service instance is created.

6. The jar file is uploaded to Server Manager and both the file and the scfjar folder are deleted from the deployment server.

Understanding the Deployment Process for WLS

This is an overview of how business services are deployed to the business services server for WLS 11g and WLS 12c:

1. When you click Deploy, the R98825F runs.

   > **Note:** If you are deploying the package to both the enterprise server and the business services server, you select the enterprise server and click Deploy. R98825D deploys the package to the enterprise server and then calls R98825F to deploy the package to the business services server.

2. The R98825F creates the scfjar folder.
3. The scf_manifest.xml is created in the scfjar folder.

   This xml contains information that the package deployment process uses to communicate with the Server Manager.

4. E1BSSVAuthenticator.jar is copied to the scfjar folder.

   E1BSSVAuthenticator.jar is a custom authenticator jar that is required to secure business services on Oracle WebLogic Server.
5. The WLS .ear file is copied to the scfjar folder.
6. The contents of the scfjar folder are combined to form the bssv_timestamp.jar file.

   Note: If errors occur, they are logged to the jas.log or jasdebug.log files that are configured through the jdelog.properties file in the E1_Install_Path\system\classes path. Errors are also logged in the Server Manager EnterpriseOne Agent logs on the machine on which WAS is installed and under which the business service instance is created.

7. The jar file is uploaded to Server Manager and both the file and the scfjar folder are deleted from the deployment server.

Prerequisites

If you are deploying JAX-WS business services to WAS v7, ensure that your WAS version is 7.0 with Fix Pack 19 and the IFIX for Fix Pack 19 (7.0.0.19-WS-WAS-IFPM62535.pak) is installed on the WAS under which the business service instance is created.

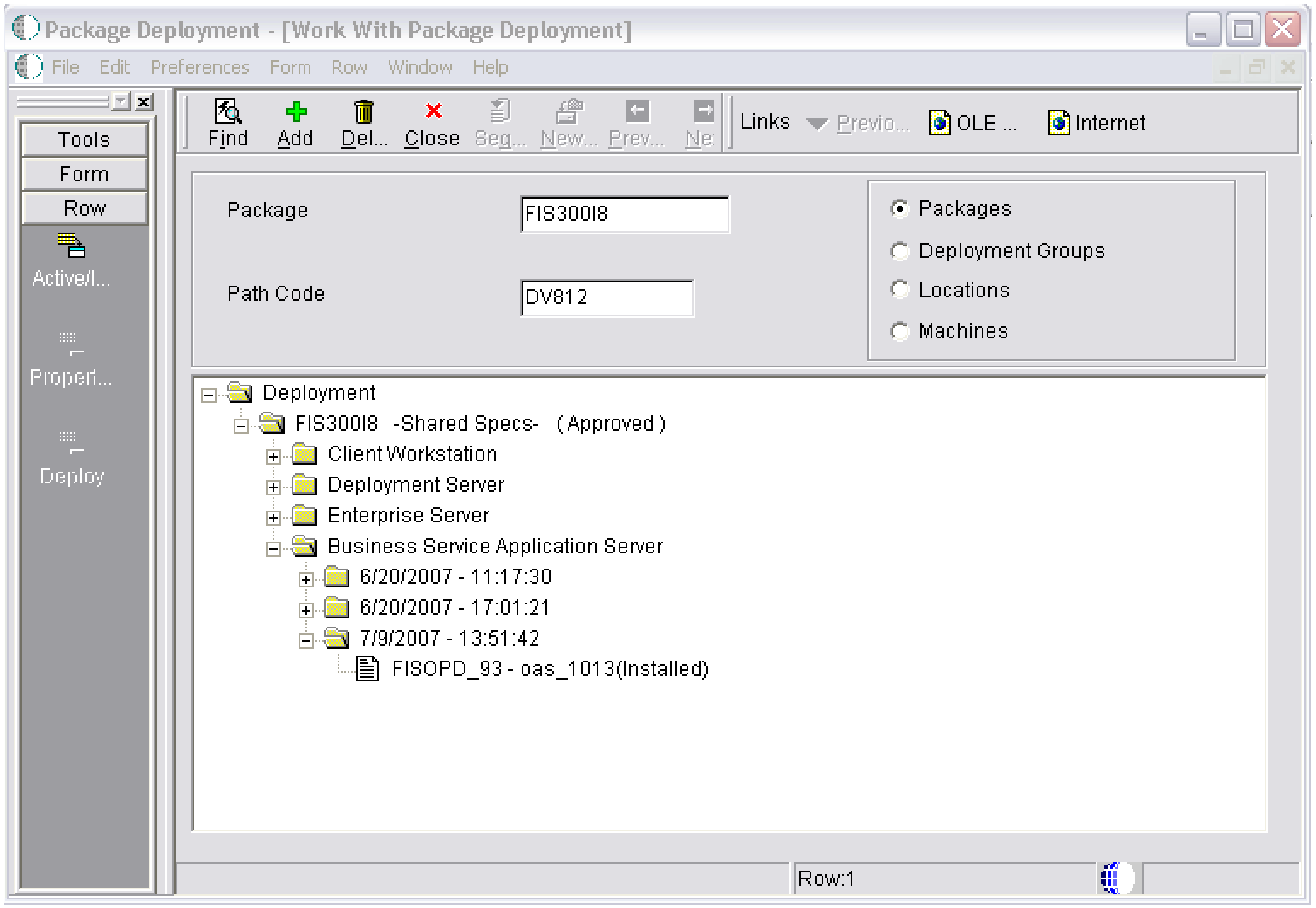
For Websphere Application server installation v8.5 or v9.0, the Web Server Plug-ins for IBM WebSphere Application Server V8.5 or V 9.0 entry in the httpd.conf has to be manually entered. The file httpd.conf is found at the location "<IBM_HTTPServer_InstallFolder>\conf\". For a windows installation an entry like the following has to be made to httpd.conf:

```
LoadModule was_ap22_module "<Plugins_Install_Folder>\bin\32bits\mod_was_ap22_http.dll"
WebSpherePluginConfig "<Plugins_Install_Folder>/config/webserver1/plugin-cfg.xml"
```

If you are deploying JAX-WS business services to WAS v8.5, ensure that the IFIX 8.5.0.0-WS-WASProd-IFPM62535 Version 8.5.0.20120907_1136 (8.5.0.20120907_1136) for WAS v8.5 is installed on the WAS under which the business service instance is created.

Deploying the Business Services

Enter P9631 in the Fast Path or go to the Package and Deployment Tools menu and select Package Deployment.



Package: FIS300I8

Path Code: DV812

(● Packages) Deployment Groups / Locations / Machines

```
Deployment
  FIS300I8  -Shared Specs-  ( Approved )
    Client Workstation
    Deployment Server
    Enterprise Server
    Business Service Application Server
      6/20/2007 - 11:17:30
      6/20/2007 - 17:01:21
      7/9/2007 - 13:51:42
        FISOPD_93 - oas_1013(Installed)
```

Row:1

1. On the Work with Package Deployment form, click the Add button.
2. On the Package Selection form, select the business services package to deploy and click Next.
3. On the Package Deployment Targets form, select Business Services Server as the Deployment Target.

> **Note:** The check box for Business Services Server is disabled if the selected package does not contain business services.

4. On the Package Deployment Attributes form, enter the Management Server URL and click Next.

   Server Manager returns a list of eligible business services servers to which the user can deploy the business services.

   > **Note:** In order to deploy the package successfully, the JD Edwards EnterpriseOne user must be a valid Server Manager user. The user cannot deploy the package if the JD Edwards EnterpriseOne user's credentials are not valid for Server Manager. The user also cannot deploy the package if there is no valid business services server defined in Server Manager.

   See the *JD Edwards EnterpriseOne Tools Server Manager Guide* for information about setting up Server Manager users.
5. Select the appropriate servers and click Next.
6. On the Business Services Package Deployment Properties Revisions form, click End.
7. On Work with Package Deployment, open your package name and then Business Service Application Server in the tree structure.
8. Select the date/time stamp and select Deploy from the Row menu.

   > **Note:** You cannot deploy to an individual business services server. Business services are deployed to all servers under the selected date/time stamp.

9 Working with Packages for Business Services (Applications Releases 9.1 and 9.0)

Understanding Packages for Business Services

After business services are created, they need to be built and deployed for consumption. The package build process builds and creates the necessary files for development client, WebLogic Server (WLS), and Websphere Application Server (WAS) consumption. The client installation process deploys business services to all development clients. The package deployment process deploys the business services archive (.ear) files to the preconfigured J2EE containers on WLS and WAS.

JD Edwards EnterpriseOne supports building JAX-WS based business service packages and JAX-RPC based business service packages.

JAX-RPC-Based Business Service Packages

Java API for XML-based Remote Procedure Calls (JAX-RPC) is an API for building standards-based web services and clients that use remote procedure calls (RPC) and XML. You can build the following combinations of JAX-RPC based business services packages:

- JAX-RPC package for WLS with JDeveloper 11g and JDeveloper 12c.
- JAX-RPC package for WAS with JDeveloper 10g and RAD.

Note: To build for WAS, you must specify the JDeveloper 10g install path in the JDeveloper Install Path field and also specify the location where IBM Rational Application Developer for WebSphere (RAD) is installed in the RAD Install Path field.

Note:

- *"Configuring Business Services Server Security" in the JD Edwards EnterpriseOne Tools Business Services Server Reference Guide* for more information on configuration and security of the Business Services Server.

JAX-WS-Based Business Service Packages

Java API for XML-based web services (JAX-WS) is an API for building standards-based web services and clients that are message oriented. JAX-WS supports transmission of SOAP messages over HTTP. JAX-WS delegates data binding-related tasks to the Java API for XML Binding (JAXB). JAXB provides support for Java to XML mapping, additional support for less used XML schema constructs, and also provides bidirectional customization of Java and XML data binding.

JD Edwards EnterpriseOne supports building the following combinations of JAX-WS based business services packages:

- JAX-WS package for WLS with JDeveloper 11g.
- JAX-WS package for WAS with JDeveloper 11g.
- JAX-WS package for WLS with JDeveloper 12c.
- JAX-WS package for WAS with JDeveloper 12c.

The JAX-WS business services package includes the JAX-WS based WSDL artifacts for the published business services and the newly developed JAX-WS business service consumer proxies. Existing JAX-RPC based business services consumer proxies are not be included in this package. If you want to use existing JAX-RPC based business services consumer proxies, then you need to build JAX-RPC based business services packages.

Note: JDeveloper 11g or 12c are required for building JAX-WS business services packages for both WLS and WAS. IBM RAD is not required for building JAX-WS business services packages for WAS. You can build JAX-WS business services packages for both WLS and WAS at the same time by selecting both the WLS and WAS options in the Business Services Assembly Application.

Note:

- *Assembling JAX-WS Based Business Services for Package Build* in this document for detailed steps.
- *"Configuring Business Services Server Security for JAX-WS Based Business Services" in the JD Edwards EnterpriseOne Tools Business Services Server Reference Guide* for more information about configuration and security of the Business Services Server.

Using IBM Rational Application Developer 7.5

You can build business services packages for a business services server running IBM WebSphere 6.1 or IBM WebSphere 7 with IBM's Rational Application Developer (RAD) 7.5.

You can build business services packages from these machines using RAD 7.5:

- JD Edwards EnterpriseOne deployment server installed on a Microsoft Windows Server 2008 R2 64-bit operating system.
- JD Edwards EnterpriseOne development client installed on a Microsoft Windows 7 operating system.

The supported version of RAD software is RAD 7.5.5 and above. It is necessary to have RAD 7.5.5 or above installed because support for Microsoft Windows Server 2008 R2 and Microsoft Windows 7 starts with this version.

You must install the following packages via IBM Installation Manager. The order of installation of the packages is not important because this is handled by the Installation Manager:

- IBM RAD for WebSphere Software 7.5.5 (or higher).
- IBM RAD Assembly and Deployment Features 7.5.5 (or higher).
- IBM WAS Version 6.1 Test Environment 6.1.0.19 (comes with the default installation). Apply the latest supported WebSphere Fix Pack listed in the certifications (Minimum Technical Requirements).

  See document 745831.1 (JD Edwards EnterpriseOne Minimum Technical Requirements Reference) on My Oracle Support: `https://support.oracle.com/epmos/faces/DocumentDisplay?id=745831.1`

Disk Space Requirements

The disk space requirements for RAD 7.5 are:

- 7.3 Giga Bytes (GB) for extracting the CDs.
- 130 Mega Bytes (MB) for IBM Installation Manager during the installation and approximately 20 MB for user data and downloads.

- 10.5 GB of disk space for IBM RAD 7.5, WebSphere 6.1 Test Environment, WebSphere 7 Test Environment, and shared folders.

Special Considerations

If you are building the business services package on the deployment server and using RAD 7.5:

- Configure H4A/WLS4A to properly work on the deployment server.
- Copy jde.ini entries specific to H4A/WLS4A from the client jde.ini (or modify and copy the entries below):

```
[LOCALWEB]
# Installation flag, if it is 0, no HTML testing setup, disable all HTML testing
AppServerInstalled=1

# Name of local web server, localhost is default but may not be valid always.
webhostname=localhost

webport=8888
webserverstart=C:\JDEdwards\E920\system\oc4j\startOC4J.bat
webserverstop=C:\JDEdwards\E920\system\oc4j\stopOC4J.bat
webserverstartarg=
webserverstoparg=
# start web server on demand, or immediately
# valid values : ONDEMAND (web server will be started on the first HTTP request) ,
                 MANUAL (web server has to be started manually by user on port
 specified),
                 IMME (web server starts as soon as ActivConsole starts)
StartAppServer=ONDEMAND

# delay time between starting web server and launching browser window
# default value is 60 (60 secs)
WebDelay=60
```

- Edit the jdbj.ini to point the environment to JDEPLAN:

```
[JDBj-BOOTSTRAP SESSION]
user=JDE
password=JDE
environment=JDEPLAN
role=*ALL
```

- Ensure that the EnterpriseOne Menu option is accessible from the Tools menu in JD Edwards Solution Explorer.

Using IBM Rational Application Developer 8.5

You can build business services packages for a business services server running on IBM WebSphere 8.5 with IBM's Rational Application Developer (RAD) 8.5 64 bit.

You can build business services packages from these machines using RAD 8.5:

- JD Edwards EnterpriseOne deployment server installed on a Microsoft Windows Server 2008 R2 64-bit operating system.
- JD Edwards EnterpriseOne development client installed on a Microsoft Windows 7 operating system.

You must install the following packages via IBM Installation Manager:

1. IBM Rational Application Developer for WebSphere Software Version 8.5 (8.5.0.RADO85-I20120529_2348).

2. IBM WebSphere Application Server V7 (64-bit) Test Environment Version 7.0.0.23 (2.0.13.WTE70-6470023-I20120517_1723). You have to download this package separately as this is not included with the default package.

> **Note:** You must first install RAD 8.5 and then install WAS v7 TE7.0.0.23 (64 bit) on top of RAD 8.5.

Disk Space Requirements

The disk space requirements for RAD 8.5 are:

- Approximately 11 GB for extracting the CDs (including the test environment).
- 150 MB for IBM Installation Manager during the installation and approximately 20 MB for user data and downloads.
- Approximately 8 GB of disk space for IBM RAD 8.5, WebSphere 7.0.0.23 Test Environment (64 bit), and shared folders.

Special Considerations

1. While installing WAS version 7.0 Test Environment 7.0.0.23 (64 Bit) on top of RAD 8.5, create the default WAS profile with the default profile name.
2. If you are building the business services package on the deployment server and using RAD 8.5:
    - Configure H4A/WLS4A to properly work on the deployment server.
    - Copy jde.ini entries specific to H4A/WLS4A from the client jde.ini (or modify and copy the entries below):

```
[LOCALWEB]
# Installation flag, if it is 0, no HTML testing setup, disable all HTML
 testing
AppServerInstalled=1

# Name of local web server, localhost is default but may not be valid always.
webhostname=localhost

webport=8888
webserverstart=C:\JDEdwards\E920\system\oc4j\startOC4J.bat
webserverstop=C:\JDEdwards\E920\system\oc4j\stopOC4J.bat
webserverstartarg=
webserverstoparg=
# start web server on demand, or immediately
# valid values : ONDEMAND (web server will be started on the first HTTP
 request) ,
                 MANUAL (web server has to be started manually by user on port
 specified),
                 IMME (web server starts as soon as ActivConsole starts)
StartAppServer=ONDEMAND

# delay time between starting web server and launching browser window
# default value is 60 (60 secs)
WebDelay=60
```

3. Edit the jdbj.ini to point the environment to JDEPLAN:

```
[JDBj-BOOTSTRAP SESSION]
user=JDE
password=JDE
environment=JDEPLAN
role=*ALL
```

4. Ensure that the EnterpriseOne Menu option is accessible from the Tools menu in JD Edwards Solution Explorer.

Assembling JD Edwards EnterpriseOne Business Services

This section lists prerequisites and discusses how to:

- Assemble JAX-RPC based business services for package build.
- Assemble JAX-WS based business services for package build.

Prerequisites

Before you complete the tasks in this section:

- Use Server Manager to create J2EE business service container(s) for the Business Services Server.
- Use Server Manager to set up Server Manager users.

  In order to deploy the package successfully, the JD Edwards EnterpriseOne user must be a valid Server Manager user. The user cannot deploy the package if the JD Edwards EnterpriseOne user's credentials are not valid for Server Manager.

  See the *JD Edwards EnterpriseOne Tools Server Manager Guide* .
- If you have multiple security servers, you must set up JD Edwards EnterpriseOne Trusted Nodes for a successful deployment of business services.

  See *"Setting Up a Trusted Node Configuration" in the JD Edwards EnterpriseOne Tools Security Administration Guide* .
- If you are building a business services package for WebLogic Server with JDeveloper 11g, JDeveloper 11.1.1.5 needs to be installed on the JD Edwards EnterpriseOne client that you are using to build the business services package.
- In the jde.ini, set the JDeveloperVersion:
  - To **11.1.1** if you are assembling a business services package for WLS 11g.
  - To **12.1.2** if you are assembling a business services package for WLS 12c.

These additional prerequisites are required if you are building a JAX-WS based business services package for WLS, WAS, or both WLS and WAS:

- Ensure that you have JDeveloper 11.1.1.5 installed on the JD Edwards EnterpriseOne client or deployment server machine that you are using to build the business services package, and set JDeveloperVersion to 11.1.1 in the jde.ini file.
- Ensure that you have JDeveloper 12.1.2 installed on the JD Edwards EnterpriseOne client or deployment server machine that you are using to build the business services package, and set JDeveloperVersion to 12.1.2 in the jde.ini file.

Assembling JAX-RPC Based Business Services for Package Build

Enter P9603 in the Fast Path.

1. On the Assemble Business Services form, in the Pathcode field, enter the path code of the package that you plan to build and tab to the next field.

   > **Note:** At this point, the application retrieves all the available business services from F98602 and F98603. If you have used this application before, the application also retrieves the values for the JDeveloper Install Path and Rational Application Developer Install Path fields from the JDE.INI file.

2. If this is your first time in the application, you must manually complete the JDeveloper Install Path and Rational Application Developer Install Path fields. Enter the following values, depending on whether you are assembling your business services to build a package for WAS or WLS:

   a. To build for WAS, enter the location where RAD is installed in the Rational Application Developer Install Path field & enter the location for the JDeveloper 10g install path in the JDeveloper Install Path field.
   b. To build for only WLS 11g, enter the location for the JDeveloper 11g install path in the JDeveloper Install Path field. The JDeveloper 11g install path should include the root folder of the JDeveloper 11g installation and not the JDeveloper folder (for example, C:\Oracle\Middleware).
   c. To build for only WLS 12c, enter the location for the JDeveloper 12c install path in the JDeveloper Install Path field. The JDeveloper 12c install path should include the root folder of the JDeveloper 12c installation and not the JDeveloper folder (for example, C:\Oracle\Middleware\Oracle_Home).
3. When you enter the install path, P9603 verifies the actual location and version.

   If this path is correct, the P9603 adds the information to the jde.ini as shown in this example:

   ```
   [MTR VALIDATION]
   JDeveloperInstallPath=<Install path specified by P9603>
   JDeveloperVersion=12.1.2
   WebSphereInstallPath=<Install path specified by P9603>
   WebSphereVersion=8.5
   ```

   > **Note:** If the path or version is incorrect, an error appears; the Close button is disabled until you enter the correct path.

4. In the grid, select the business services that you want to expose as a web service and click the Select button.

   You can also double-click the row headings of the business services that you want to expose.

   A check mark appears by each business service that is selected.
5. Click Select again or double-click the row header to unexpose the web service.
6. Click Close to close the application.

Assembling JAX-WS Based Business Services for Package Build

Enter P9603 in the Fast Path.

1. On the Assemble Business Services form, select the JAX-WS option.

   The WLS and WAS options are enabled.

2. Do one of the following:
   - Select the WLS option and provide the JDeveloper 11.1.1.5 install path to build for WLS 11g.
   - Select the WLS option and provide the JDeveloper 12c install path to build for WLS 12c.
   - Select the WAS option and provide the JDeveloper 11.1.1.5 or JDeveloper 12c install path to build for WAS.

     > **Note:** The package for WAS built with JDeveloper 12c is supported only on a WAS 8.5 container hosted with JDK 1.7, or beginning with Tools Release 9.2.1.0, it is also supported on a WAS 9 container hosted with JDK 1.8.

   - Select both WLS and WAS options and provide the JDeveloper 11.1.1.5 install path to build for WLS 11g, and WAS 7 or 8.5.
   - Select both WLS and WAS options and provide the JDeveloper 12c install path to build for both WLS 12c and WAS 8.5 with JDK 1.7 or (Tools Release 9.2.1.0) WAS 9 with JDK 1.8.

     The JAX-WS EAR for WLS is built first, and then the JAX-WS based EAR for WAS is built.
3. In the Pathcode field, enter the path code of the package that you plan to build and tab to the next field.

   > **Note:** At this point, the application retrieves all the available business services from F98602 and F98603. If you have used this application before, the application also retrieves the values for the JDeveloper Install Path field from the JDE.INI file.

4. If this is your first time in the application, you must manually enter the JDeveloper install path (on the package build machine) in the JDeveloper Install Path field.

   The JDeveloper 11g install path should include the root folder of the JDeveloper 11g installation (for example, C:\Oracle\Middleware), and not the JDeveloper folder.

   The JDeveloper 12c install path should include the root folder of the JDeveloper 12c installation (for example, C:\Oracle\Middleware\Oracle_Home), and not the JDeveloper folder.
5. When you enter the install path, P9603 verifies the actual location and version.

   If this path is correct, P9603 adds this information to the jde.ini:

   ```
   [MTR VALIDATION]
   JDeveloperInstallPath=<Install path specified by P9603>
   ```

   > **Note:** If the path or version is incorrect, an error appears; the Close button is disabled until you enter the correct path.

6. The RAD Install Path field is disabled because RAD is not required for building JAX-WS business services packages for WAS.
7. In the grid, select the business services that you want to expose as a web service and click the Select button.

   You can also double-click the row headings of the business services that you want to expose.

   A check mark appears by each business service that is selected.
8. Click Select again or double-click the row header to hide the web service.
9. Click Close to close the application.

Understanding the Assemble Business Services Form

After you set the JDeveloperVersion to 11.1.1 in the jde.ini file, the Assemble Business Services form shows the JAX-RPC and JAX-WS options. You use this version of the form to build JAX-WS based business services packages for WLS, WAS, or both WLS and WAS.

You can use this version of the form to build JAX-RPC based business services packages for WLS. To assemble a JAX-RPC based business services package for WLS, first select the JAX-RPC option, and then enter the location for the JDeveloper 11g install path in the JDeveloper Install Path field, as illustrated here:

Assembling a Package that Contains Published Business Services

This section discusses how to assemble a business service package.

Assembling a Business Service Package

To set the processing options for Package Assembly, go to the Package and Deployment Tools menu, right-click the Package Assembly application (P9601), and select prompt for values.

1. Set the processing option entitled Business Services to **1.** This processing option is blank by default.
2. Select OK.
3. On the Work with Packages form, begin the assembly process.

   See *Assembling Packages*.

Building a Package with Published Business Services

This section provides overviews of the JAX-RPC and JAX-WS build process, prerequisites, and discusses how to:

- Define a package build with published business services.
- Resubmit the package build.

Understanding the Build Process for a JAX-RPC Based Business Service Package

This section provides overviews of how the JD Edwards EnterpriseOne system builds a package that contains business services for WAS or for WLS.

Build Process for WAS

For WAS, the JD Edwards EnterpriseOne system:

1. Creates the \\work\sbf\sbfbuild.ini, which defines the paths to the exposed methods.
2. Creates the Ant scripts logtimestamp.xml and build.xml in the \\work\sbf directory.
3. Runs the build.xml Ant script to extract source.

   When the extract occurs, Unjar_BusinessService.log is generated in the \\work\sbf directory.
4. Creates service interface files.

   An interface file is created for each published business service. The selected methods are listed in the created published business service.

5. Creates the Web Service Inspection Language (WSIL) file, which is used for Business Process Execution Language (BPEL).
6. Creates Ant scripts for WAS.

   These scripts are created within the \\work\sbf\WAS directory.
7. Runs the WAS build.xml Ant script.

   The build.xml script creates an .ear file that is WAS-compatible.
8. Compresses the java folder for deployment of the client sbf.cab.

   Note: Review the was_BusinessService.log to verify that the build was successful. If the build is successful, "Build Successful" appears at the bottom of the log. If the build failed, "Build Failed" appears at the bottom of the log.

Build Process for WLS

For WLS, the JD Edwards EnterpriseOne System:

1. Creates the \\work\sbf\sbfbuild.ini, which defines the paths to the exposed methods.
2. Creates the Ant scripts, logtimestamp.xml and build.xml in the \\work\sbf directory.

   Note: For WLS 12c, Ant is Version 1.7.
3. Runs the build.xml Ant script to extract source. When the extract occurs, the Unjar_BusinessService.log is generated in the \\work\sbf directory.
4. Creates the Java Web Service (JWS) files. A JWS file is created for each published business service. The selected methods are listed in the created published business service.
5. Creates the WSIL file, which is used for BPEL.
6. Creates Ant scripts for WLS. These scripts are named build.properties and build.xml. They are created within the \\work\sbf\wls directory.
7. Runs the WLS build.xml Ant script.

   The build.xml Ant script:

   a. Creates javadoc.
   b. Compiles java source files.
   c. Assembles the business services' source into an .ear file that is WLS-compatible.
8. Compresses the java folder for deployment of the client sbf.cab.

Note: Review the wls_buildservices.log in \\work\sbf\wls to verify that the build was successful. If the build is successful, "Build Successful" appears at the bottom of the log. If the build failed, "Build Failed" appears at the bottom of the log.

Understanding the Build Process for a JAX-WS Based Business Service Package

This section provides overviews of how the JD Edwards EnterpriseOne system builds a package that contains JAX-WS based business services for WLS & WAS.

For building a JAX-WS based business services package for WLS & WAS, the JD Edwards EnterpriseOne System:

1. Creates the \\work\sbf\sbfbuild.ini, which defines the paths to the exposed methods.
2. Creates the Ant scripts, logtimestamp.xml and build.xml, in the \\work\sbf directory.
3. Runs the build.xml Ant script to extract source.

   When the extract occurs, Unjar_BusinessService.log is generated in the \\work\sbf directory.
4. Creates Java Web Service (JWS) files.

   The JWS files are created on the EnterpriseOne Deployment Server in the E1_Install_Path\Pathcode \Package\PackageName\java\sbf\sbfjaxwswls directory for WLS and at E1_Install_Path\Pathcode\Package \PackageName\java\sbf\sbfjaxwswas directory for WAS.

   A JWS file is created for each published business service, and it has annotations that describe the published business service. The selected methods are listed in the created published business service. The JWS files are required to create the WSDL artifacts for the published business services, and they should not be edited manually.
5. Creates the Web Service Inspection Language (WSIL) file, which is used for Business Process Execution Language (BPEL).
6. Creates Ant scripts for WLS and WAS.

   These scripts are named build.compile.xml, build.xml, and sbfwebservices.xml. They are created within the \ \work\sbf\wls directory for WLS and within the \\work\sbf\was directory for WAS.
7. Runs the build.xml Ant script.

   The build.xml Ant script:

   a. Creates javadoc.
   b. Compiles the business services java source files.
   c. Invokes the sbfwebservices.xml Ant script to run the wsgen Ant task

      Invokes the sbfwebservices.xml Ant script to run the wsgen Ant task to create the web services artifacts (such as the WSDL file).
   d. Generates the required deployment descriptors, and assembles the business services classes into an .ear file that is compatible with WLS and WAS.

When the build process finishes, an E1Services-Pathcode-wls.ear file is created for WLS, and an E1Services-Pathcode-was.ear file is created for WAS on the EnterpriseOne Deployment Server within the E1_Install_Path\Pathcode\Package \PackageName directory.

Note: For WLS, review the wls_BusinessService.log in the \\work\sbf\wls directory and for WAS, review the was_BusinessService.log in the \\work\sbf\wls directory to verify that the build was successful. If the build is successful, *Build Successful* appears at the bottom of the log. If the build failed, *Build Failed* appears at the bottom of the log.

Prerequisites

Before building the package, verify that logging is turned off. When the jdeproperties.log file is set for logging in the \system\classes folder and a package build is submitted, the build process is slowed down. It is not recommended to have logging turned on during package builds.

Defining a Package Build with Published Business Services

To define a package build with published business services:

1. Enter P9621 in the Fast Path or go to the Package and Deployment Tools menu and select Package Build.
2. Use the Package Build application to define build properties for the package.
3. On the Package Build Location form, select Client as the Build Location.

> **Note:** Business services are not built for server-only packages.

The JVM's virtual memory is set to a maximum of 512 MB. You can change this using the following jde.ini settings:

```
[PACKAGE BUILD]

JavacMaxMemorySize= 512 MB

JavadocMaxMemorySize= 512 MB
```

> **Note:** The javadoc task with JDeveloper 12c (JDK 1.7) requires more memory to load the business services and to create javadoc during the business service build process for both JAX-RPC and JAX-WS. The following error can occur if you use the default memory of 512 MB:
>
> ```
> javadoc: error - java.lang.OutOfMemoryError: Please increase memory.
> ```
>
> When building business services packages with JDeveloper 12c, increase the JDE.INI entry **JavadocMaxMemorySize** to 760 MB.

Resubmitting the Package Build

If errors occur during the package build, you will need to reset the status and resubmit the package build.

Enter P9622 in the Fast Path or go to the Package and Deployment Tools menu and select Package Build History.

1. On the Work with Package Build History form, enter your package in the Package Name field and select Find.
2. In the tree structure, expand your package name and CLIENT.
3. Click Business Services and select your business service.
4. Select Reset Status from the Row menu to reset the status of your business services.
5. Select Resubmit Build from the Row menu.

Deploying the Package to the Business Services Server

This section provides overviews of the deployment process for WAS and WLS, and discusses how to deploy the business services.

Understanding the Deployment Process for WAS

This is an overview of how business services are deployed to the Business Services Server for WAS.

1. When you click Deploy, the R98825F runs.

   > **Note:** If you are deploying the package to both the enterprise server and the business services server, you select the enterprise server and click Deploy. R98825D deploys the package to the enterprise server and then calls R98825F to deploy the package to the business services server.

2. The R98825F creates the scfjar folder.
3. The scf_manifest.xml is created in the scfjar folder.

   This xml contains information that the package deployment process uses to communicate with the Server Manager.
4. The WAS .ear files are copied to the scfjar folder.
5. The contents of the scfjar folder are combined to form the bssv_timestamp.jar file.

   > **Note:** If errors occur, they are logged to the jas.log or jasdebug.log files that are configured through the jdelog.properties file in the E1_Install_Path\system\classes path. Errors are also logged in the Server Manager EnterpriseOne Agent logs on the machine on which WAS is installed and under which the business service instance is created.

6. The jar file is uploaded to Server Manager and both the file and the scfjar folder are deleted from the deployment server.

Understanding the Deployment Process for WLS

This is an overview of how business services are deployed to the business services server for WLS 11g and WLS 12c:

1. When you click Deploy, the R98825F runs.

   > **Note:** If you are deploying the package to both the enterprise server and the business services server, you select the enterprise server and click Deploy. R98825D deploys the package to the enterprise server and then calls R98825F to deploy the package to the business services server.

2. The R98825F creates the scfjar folder.
3. The scf_manifest.xml is created in the scfjar folder.

   This xml contains information that the package deployment process uses to communicate with the Server Manager.
4. E1BSSVAuthenticator.jar is copied to the scfjar folder.

   E1BSSVAuthenticator.jar is a custom authenticator jar that is required to secure business services on Oracle WebLogic Server.
5. The WLS .ear file is copied to the scfjar folder.

6. The contents of the scfjar folder are combined to form the bssv_timestamp.jar file.

   Note: If errors occur, they are logged to the jas.log or jasdebug.log files that are configured through the jdelog.properties file in the E1_Install_Path\system\classes path. Errors are also logged in the Server Manager EnterpriseOne Agent logs on the machine on which WAS is installed and under which the business service instance is created.

7. The jar file is uploaded to Server Manager and both the file and the scfjar folder are deleted from the deployment server.

Prerequisites

If you are deploying JAX-WS business services to WAS v7, ensure that your WAS version is 7.0 with Fix Pack 19 and the IFIX for Fix Pack 19 (7.0.0.19-WS-WAS-IFPM62535.pak) is installed on the WAS under which the business service instance is created.

For Websphere Application server installation v8.5 or v9.0, the Web Server Plug-ins for IBM WebSphere Application Server V8.5 or V 9.0 entry in the httpd.conf has to be manually entered. The file httpd.conf is found at the location "<IBM_HTTPServer_InstallFolder>\conf\". For a windows installation an entry like the following has to be made to httpd.conf:

```
LoadModule was_ap22_module "<Plugins_Install_Folder>\bin\32bits\mod_was_ap22_http.dll"
WebSpherePluginConfig "<Plugins_Install_Folder>/config/webserver1/plugin-cfg.xml"
```

If you are deploying JAX-WS business services to WAS v8.5, ensure that the IFIX 8.5.0.0-WS-WASProd-IFPM62535 Version 8.5.0.20120907_1136 (8.5.0.20120907_1136) for WAS v8.5 is installed on the WAS under which the business service instance is created.

Deploying the Business Services

Enter P9631 in the Fast Path or go to the Package and Deployment Tools menu and select Package Deployment.

1. On the Work with Package Deployment form, click the Add button.
2. On the Package Selection form, select the business services package to deploy and click Next.
3. On the Package Deployment Targets form, select Business Services Server as the Deployment Target.

   Note: The check box for Business Services Server is disabled if the selected package does not contain business services.

4. On the Package Deployment Attributes form, enter the Management Server URL and click Next.

   Server Manager returns a list of eligible business services servers to which the user can deploy the business services.

   > **Note:** In order to deploy the package successfully, the JD Edwards EnterpriseOne user must be a valid Server Manager user. The user cannot deploy the package if the JD Edwards EnterpriseOne user's credentials are not valid for Server Manager. The user also cannot deploy the package if there is no valid business services server defined in Server Manager.

   See the *JD Edwards EnterpriseOne Tools Server Manager Guide* for information about setting up Server Manager users.
5. Select the appropriate servers and click Next.
6. On the Business Services Package Deployment Properties Revisions form, click End.
7. On Work with Package Deployment, open your package name and then Business Service Application Server in the tree structure.
8. Select the date/time stamp and select Deploy from the Row menu.

   > **Note:** You cannot deploy to an individual business services server. Business services are deployed to all servers under the selected date/time stamp.

10 Harvesting Published Business Services into the Oracle Enterprise Repository Server

Overview

The information in this chapter applies only to customers using JD Edwards EnterpriseOne business services and Oracle Enterprise Repository.

JD Edwards EnterpriseOne provides the ability to harvest information about published business services to the Oracle Enterprise Repository. Oracle Enterprise Repository is a metadata repository that serves as a single source of truth for information about Oracle Service-Oriented Architecture (SOA) assets and their dependencies. Oracle Enterprise Repository is a searchable repository that manages assets and relationships between assets.

Prerequisites

Before harvesting JD Edwards EnterpriseOne published business services into the Oracle Enterprise Repository server, perform the following prerequisites:

- Set up the Oracle Enterprise Repository server.

  For information about the installation and setup of the Oracle Enterprise Repository server, see *Oracle® Fusion Middleware Installation Guide for Oracle Enterprise Repository*.

  http://docs.oracle.com/cd/E23943_01///doc.1111/e15745/title.htm

- Configure the Harvester tool.

  The Harvester is used to harvest JD Edwards EnterpriseOne published business services artifacts into the Oracle Enterprise Repository server.

  The Harvester is included in the Oracle Enterprise Repository 11g installation in the 11.1.1.3.0-OER-Harvester.zip file. The .zip file is located in the following Oracle Enterprise Repository installation directory:

  ```
  <ORACLE_HOME>/repositoryXXX/core/tools/solutions/11.1.1.3.0-OER-Harvester.zip
  ```

  To configure the Harvester:

  > **Note:** The following steps are general and should work with most configurations. For detailed instructions on configuring the Harvester, see "Configuring and Using Automated Harvesting in Design-time and Runtime Environments" in the *Oracle® Fusion Middleware Configuration Guide for Oracle Enterprise Repository.*
  > ***http://docs.oracle.com/cd/E28280_01///admin.1111/e16580/title.htm***

  a. Extract the 11.1.1.3.0-OER-Harvester.zip to a <Harvester Home> directory.
  b. Set the JAVA_HOME environment variable to the JDK 1.6 home.
  c. In the Harvester subfolder, open the HarvesterSettings.xml file.
  d. In the HarvesterSettings.xml file, under the repository section, complete the following elements:

- <uri>.

  Enter the URL to the Oracle Enterprise Repository server.
- <user> (in the credentials section).

  Enter the Oracle Enterprise Repository admin user.
- <password> (in the credentials section).

  Enter the Oracle Enterprise Repository admin password.

e. Access encryptRun encrypt.bat to encrypt the password stored in the HarvesterSettings.xml file.

   The encrypt.bat is located in the Harvester subfolder.

f. Use the following command to run encrypt.bat:

```
encrypt.bat HarvesterSettings.xml HarvesterSettings.xml
```

Generating Business Service Asset Definition XML Files/ Artifacts

To harvest JD Edwards EnterpriseOne published business services artifacts into the Oracle Enterprise Repository server, you have to configure the business services package build process to generate one business service asset definition XML file per published business service (selected in the Assemble Business Services application).

An asset definition XML file contains information about the published business service, such as the name of the published business service, the published method names, the web services description language (WSDL) location of the published business service, and so forth.

To generate the business service asset definition XML files during the business services package build process, add the following parameter and value to the MTR VALIDATION section of JD Edwards EnterpriseOne jde.ini file for the Microsoft Windows client:

```
GenerateAssetXml=1
```

Note: By default, the GenerateAssetXml flag is not present in the JD Edwards EnterpriseOne Windows client jde.ini file. Therefore, the asset definition XML files are not generated by default during the business services package build process.

The following example shows an asset definition .xml file generated for the RI_CustomerManager published business service:

```
<?xml version="1.0" encoding="UTF-8"?>
<AssetDef xmlns="http://xml.oracle.com/AIA/CAR/V1" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://xml.oracle.com/AIA/CAR/V1 auasset.xsd">
<Instance>
<Core>
<UID>
<AssetType>JDEdwardsE1: Business Service</AssetType>
<TargetNameSpace>http://xml.oracle.com/JDEdward/EnterpriseOne</TargetNameSpace>
<Name>JPR01020.RI_CustomerManager</Name>
<Version>E812</Version></UID>
```

```
<Description>RI_CustomerManager</Description>
<LocationURL>Actual WSDL URL of PBSSV to be set here after BSSV Package Deploy</
LocationURL>
<Dynamic>
<Name>getCustomerCreditInfo</Name>
<Value>Published method for getCustomerCreditInfo retrieves credit information for
 Customer using provided entity id.
       Acts as wrapper method, passing null context and null connection, will call
 protected getCustomerCreditInfo.
       This is a reference implementation for getting customer credit information, it
 should not be used in Production.
</Value>
</Dynamic>
<Category>
<ApplicationName>JD Edwards EnterpriseOne</ApplicationName>
<Keywords/></Category></Core>
<Extend>
<Category>
<Name/>
<Value/></Category>
<Attribute>
<Name>Application Release</Name>
<Value>E812</Value></Attribute>
<Attribute>
<Name>Production Code</Name>
<Value>H95</Value></Attribute>
<Attribute>
<Name>Production Code Description</Name>
<Value>Object and Environment Tech</Value></Attribute>
<Attribute>
<Name>Published Business Service Class</Name>
<Value>oracle.e1.bssv.JPR01020.RI_CustomerManager</Value></Attribute>
<Attribute>
<Name>Java Doc Location</Name>
<Value>rep://JDEdwardsE1: Business Service:DV812_JAVA_DOC/oracle/e1/bssv/JPR01020/
RI_CustomerManager.html</Value></Attribute>
<Attribute>
<Name>Java Doc URL</Name>
<Value>click here for Java Doc for RI_CustomerManager</Value>
</Attribute>
</Extend>
</Instance>
</AssetDef>
```

When the business services package build process completes successfully, the generated asset definition XML files are placed on the deployment server at:

DEP_SERVER\APP_RELEASE\PATHCODE\PACKAGE\PACKAGENAME\ASSETDEFNFILES

Understanding the LocationURL Element in the Asset Definition XML File

After the business services package build process completes successfully and the asset definition xml files are generated, the business services package must be deployed to a valid business services server instance.

After a successful business services package deployment, the LocationURL element in each of the business service asset definition xml files is automatically updated with the actual WSDL URL of the deployed published business service.

For business services deployed to WebSphere Application Server (WAS), the WSDL URL is an http-based URL. For example, the LocationURL element for the RI_CustomerManager published business service that is deployed to WAS is set as:

```
<LocationURL>http://MachineName:HTTPPortNo /DV812_WEB/services/
RI_AddressBookManagerHttpPort/wsdl/RI_AddressBookManager?WSDL</LocationURL>
```

For business services deployed to Oracle WebLogic Server, the WSDL URL is specified as an https-based URL. For example, the LocationURL element for the RI_CustomerManager published business service deployed to WebLogic Server is set as:

```
<LocationURL>https://MachineName:SSL/TLSPortNo /DV812/RI_CustomerManager?WSDL</
LocationURL>
```

When the business services package is deployed to the WebLogic Server, SSL/TLS (Transport Layer Security) is enabled and the SSL/TLS Listen Port is set as the HTTP Listen Port of the Business Services Server instance on WebLogic Server plus 1. For example, when the Business Services Server instance is created on WebLogic Server, if the HTTP listen port for the instance is 7771, then during the deploy process the SSL/TLS Listen Port is set to 7772. This same value is set in the LocationURL element of the asset definition xml file for the published business service.

Note: If you manually change the SSL/TLS Listen port of the Business Services Server instance on WebLogic Server after the business services package is deployed, then you must re-deploy the business services package to the same instance in order for this new SSL/TLS Listen port to be reflected in the LocationURL element of the asset definition xml files for the published business services.

Harvesting the Business Service Asset Definition XML Files/Artifacts into the Oracle Enterprise Repository Server

To harvest the business service assets into the Oracle Enterprise Repository server:

1. In the Harvester, specify the Harvester configuration file and the folder where the generated business service asset definition xml files are located.
2. Enter the following command to direct the Harvester to harvest the business service assets to the Oracle Enterprise Repository server:

   ```
   CARHarvest -settings .\Harvester\HarvesterSettings.xml -input
   c:\bssv_xml
   ```

   Note: HarvesterSettings.xml is the Harvester configuration file that has already been configured. As an input, you must provide the folder where the generated business service asset definition XML files are located. In the preceding command, c:\bssv_xml is the folder that contains all of the JD Edwards EnterpriseOne published business service asset definition xml files.

Configuring Java Doc Location in Oracle Enterprise Repository for the Published Business Services

Each JD Edwards EnterpriseOne business service asset has a link to the Java Doc of its Java published business service class. After a business services package build is complete, you can obtain the Java Doc for the published business service classes from the following location:

```
dep_server\app_release\pathcode\package\package_name\java\sbf\javadoc
```

The Java Doc can also be extracted from the sbf.cab file which is generated during the full business services package build process.

Using an Oracle Enterprise Repository supported transport protocol, such as HTTP or FTP, configure the Java Doc location in the Oracle Enterprise Repository server to point to the actual Java Doc location of the Java published business service class. To do so, perform the following steps:

1. Make the Java Doc html files for the published business services available on the web through an HTTP server or FTP server.
2. To configure the Java Doc server mapping in Oracle Enterprise Repository to point to the HTTP or FTP server that hosts the Java Doc html files, log in to Oracle Enterprise Repository server.
3. Click the Edit/Manage Assets link to open the Asset Editor window.
4. From the Actions menu, click Configure Artifact Stores.
5. Click Add to add a new Artifact Store.
6. On Create a new Artifact Store, complete the following fields:
   - Name

     Enter the name for the artifact store. For example:

     `JDEDWARDSE1: BUSINESS_SERVICE:`*`DV812`*`_JAVA_DOC`

     Where DV812 is the JD Edwards EnterpriseOne path code.
   - Type

     Select a transport type, such as HTTP.
   - Hostname

     Enter the server name and port, for example: **`dnshravindvm1.mlab.jdedwards.com:80`**.
   - Path

     Enter the path to the Java Doc html files.
7. Click OK to add the Java Doc server mapping.

   A link to the Java Doc is established for every business service asset in the Oracle Enterprise Repository server.

Troubleshooting the Business Services Package Build and Deployment Process for Harvesting Published Business Services Artifacts

This section provides troubleshooting tips for harvesting published business services artifacts into the Oracle Enterprise Repository server.

Turn on Logging for Business Services Package Build

Before you build the business services package, activate the jas.log and jasdebug.log files. To do so, on the machine on which you are performing the business services package build, open the jdelog.properties file located in the *EnterpriseOneInstallPath*/system/classes folder and enter the appropriate value to activate the log files.

Business Service Asset Definition XML Files Not Generated

If the business service asset definition XML files are not generated, refer to the ClientPkgBuild.log on the deployment server and the jas.log and jasdebug.log files on the business services package build machine.

If the business service asset definition XML files are not updated properly, refer to jas.log and jasdebug.logs on the business services package build machine and SM e1_agent.logs on the Business Services Server.

11 Setting Up Multitier Deployment

Understanding Multitier Deployment

This section discusses:

- Overview of multitier deployment.
- Multitier deployment terminology.
- Multitier deployment features.
- Multitier implementation.
- Multitier deployment case study.

Overview of Multitier Deployment

JD Edwards EnterpriseOne software is normally distributed to workstations from a centralized location. In many cases, you can minimize the affect on the performance of a single deployment server by using systematic scheduling for software installations. For example, if your site has more than 50 workstations that require a package installation but you release software only four times a year, you can mitigate performance problems by scheduling the installations over a weekend, at night, or during off-peak hours.

While this distribution method is the simplest approach for software deployment, network capacity constrains configurations that have either multiple remote sites or large numbers of users at a single site. For example, software installations to workstations that are connected to the centralized deployment location by a 56 KB circuit can take 4 to 6 hours to run.

Multitier deployment provides sites the flexibility of installing packages on workstations and servers from more than one deployment location and more than one deployment server. These additional deployment locations and servers are called deployment tiers. Specifically, instead of installing multiple workstations across a wide area network (WAN) circuit, multitier deployment enables you to transfer a compressed package from the centralized location to the remote workgroup server, which acts as a second deployment tier. Multitier deployment means deploying from more than one deployment tier.

For example, you might have one deployment server at the main location and a second deployment server for a remote location. Because the server at the remote location is responsible for deploying to workstations and servers at that location, you do not need to deploy packages from the main deployment server across a WAN, as you would in a single-tier deployment configuration.

Workgroup servers can also be used as second-tier deployment locations in a local area network (LAN) environment, where large numbers of workstations need to install software concurrently. It is recommended that you implement multitier deployment if your site has more than 50 workstations receiving multiple installations per day.

The primary function of multitier deployment is to reduce network traffic (and the delays that result from heavy traffic) by enabling enterprises with more than one geographic location to deploy from a secondary deployment server at the remote site. Instead of installing packages across the WAN from the deployment server to workstations at a remote location, you can copy the package and the package.inf file from the deployment server at the primary location to the deployment server at the remote location. The server at the remote location can then deploy the package to the workstations and servers at that location.

Consider implementing a multitier deployment configuration when either of these situations applies:

- Too many workstations are installing packages from the same location, causing server and network performance to suffer.
- Workstations are remotely connected to the deployment server over a WAN, which requires unacceptably long installation times.

Normally, you decide whether to implement multitier deployment during Configurable Network Computing (CNC) implementation. Although you can enable this function at any time, you typically set it up after you have installed the software and are preparing the production sites to go live.

To set up multitier deployment, you must define the machines (and their associated path codes) that are used for deployment. In addition, you can use a scheduler function to define when software should be pushed to the tiered deployment location.

You must also define individual user characteristics for multitier deployment. Normally, you do this by modifying the user profiles to indicate the deployment location from which a user pulls a package.

Multitier Deployment Terminology

This table lists and describes multitier deployment terms:

Term	Description
Primary Deployment Location (tier 1)	The primary or base deployment location is the location in which the package to be deployed to secondary or remote locations is built. The system requires at least one deployment server for installing and maintaining the software. The server at the primary deployment location should be dedicated solely to deploying and operating JD Edwards EnterpriseOne products, and should not be used for any other purposes in your company.
Tier Deployment Location (tier 2)	Tier deployment locations (also known as remote or secondary locations) have one or more deployment servers that enable you to install the software on the workstations at that location. These servers receive their packages from the deployment server at the primary or base location. Machines at the tier deployment locations cannot use Object Librarian functions such as object check-in and check-out. These machines are designed for deployment use only and are not designed to be used for remote development. The number of tiered locations that you can have depends on the network and server capacity.
Tier Workstations	Tier workstations are workstations that connect to a tier deployment location for software installation. The number of workstations per deployment location depends on the network and machine load. All tiered workstations must also have a Microsoft Windows domain connection that enables them to connect to, read, and copy files from the shared drives of their respective deployment locations.

Multitier Deployment Features

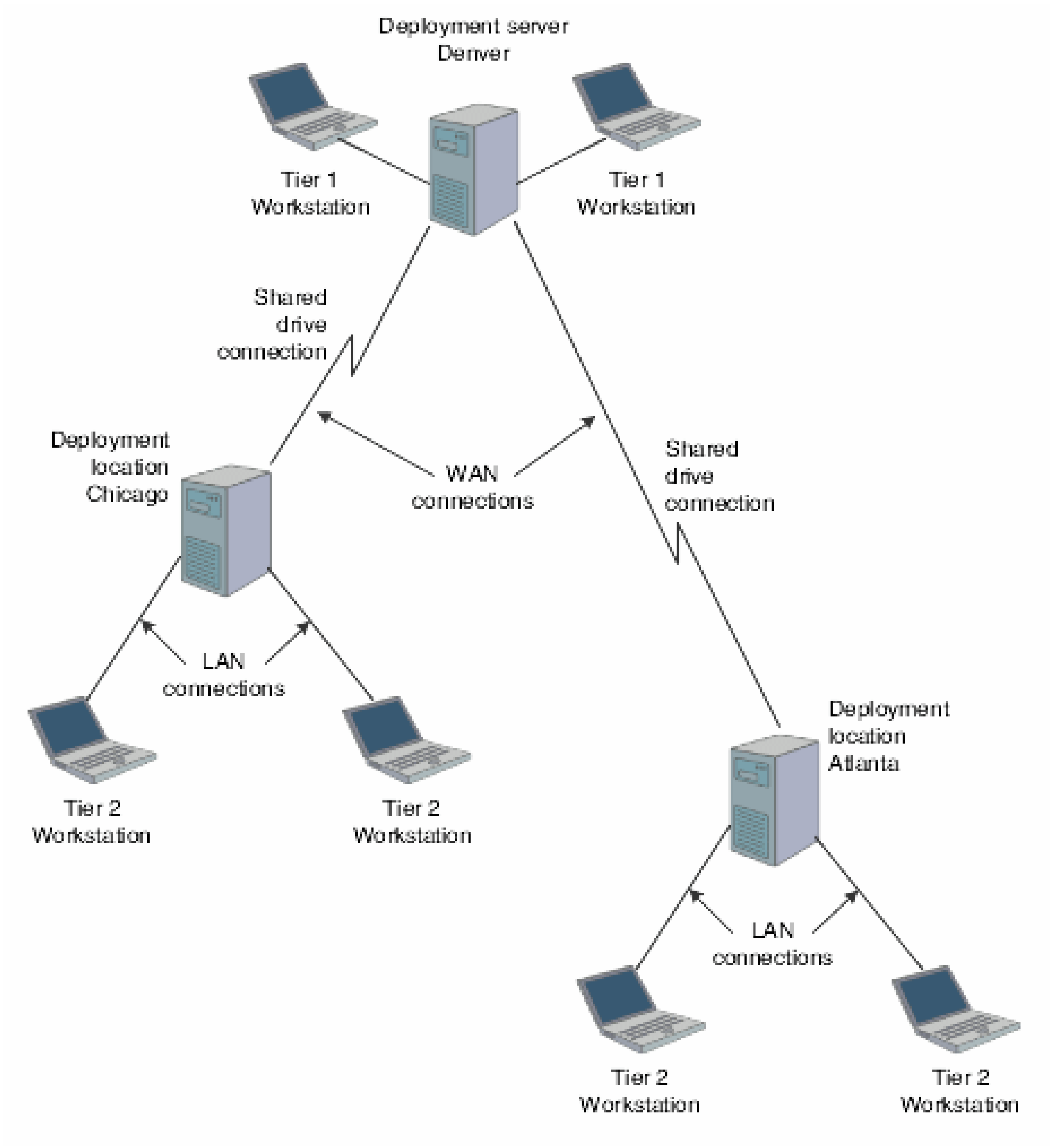
Multitier deployment offers these features:

- You can deploy workstations from any number of deployment servers.

- You can easily add deployment locations, and the deployment machine does not need to be a server; it can be a Microsoft Windows workstation.
- Setup and administration are straightforward tasks.
- You maintain central control over files and data that are transferred to remote deployment locations.
- You can easily schedule software for deployment to remote sites.
- Multitier deployment is integrated into Oracle's JD Edwards EnterpriseOne Deployment Director, so the process for deploying is essentially the same as the process for deploying in a single-tiered setup.

Example: Two-Tier Deployment Strategy

This diagram illustrates a typical two-tier deployment strategy:


Deployment server
Denver
Tier 1
Workstation
Tier 1
Workstation
Shared
drive
connection
Deployment
location
Chicago
WAN
connections
Shared
drive
connection
LAN
connections
Tier 2
Workstation
Tier 2
Workstation
Deployment
location
Atlanta
LAN
connections
Tier 2
Workstation
Tier 2
Workstation

Multitier Implementation

Packages are always built on the deployment server at the primary location. After you build the package that you want to deploy to remote locations, you must follow these steps to implement multitier deployment:

1. Define deployment locations.

   You must define each physical location to which you want to deploy. For example, if the main office is in Denver and you want to deploy to the branch office in Seattle, you must define the Seattle deployment location.
2. Create deployment server definitions.

   You must define the deployment server at each remote location.

   Note: This step is not necessary if you used Oracle's JD Edwards EnterpriseOne Remote Location Workbench to create deployment server definitions when you installed the software.
3. Schedule the package.

   Use the JD Edwards EnterpriseOne Deployment Director program (P9631) to schedule the package for deployment. The process of scheduling a package for multitier deployment is identical to the process for scheduling any other package.
4. Deploy the package to workstations.

   After you deploy the package to the deployment server at the remote location, that server can deploy to the workstations at that location.

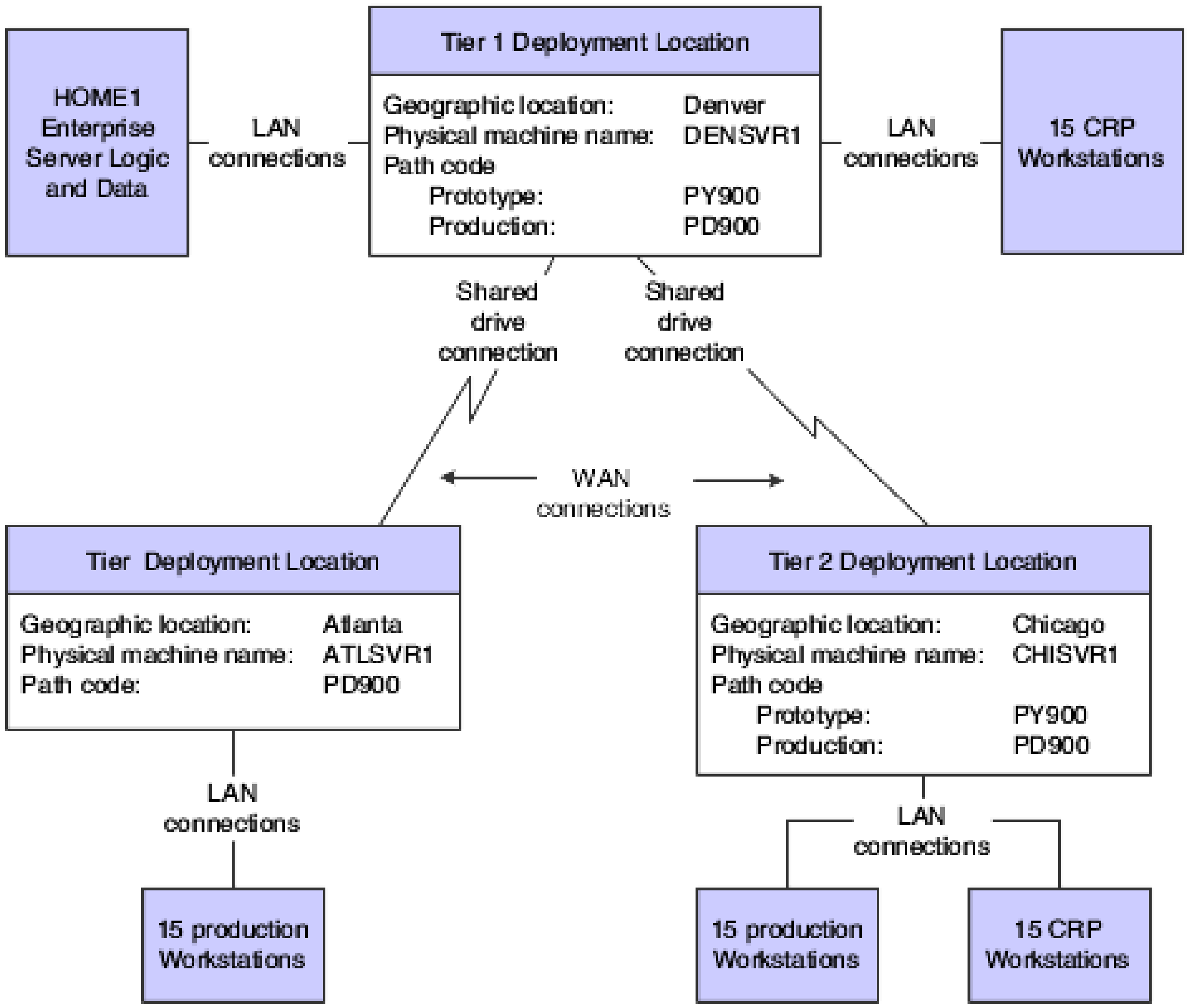
Multitier Deployment Case Study

This case study is for a two-tier deployment environment. As this case study demonstrates, deploying packages to workstations using WAN connections is generally not efficient. Instead, you should deploy from a primary deployment server to tier deployment locations. After that, you can install packages to LAN-attached workstations from each local deployment location.

For package installations, a remote deployment location functions as a file server. You cannot build packages at a remote deployment location; packages must be built at the primary deployment location.

While locally attached workstations can pull packages from the tier deployment location, these workstations still require enterprise server and database server connectivity.

This diagram illustrates this case study:


HOME1 Enterprise Server Logic and Data
LAN connections
Tier 1 Deployment Location
Geographic location: Denver
Physical machine name: DENSVR1
Path code
Prototype: PY900
Production: PD900
LAN connections
15 CRP Workstations
Shared drive connection
Shared drive connection
WAN connections
Tier Deployment Location
Geographic location: Atlanta
Physical machine name: ATLSVR1
Path code: PD900
Tier 2 Deployment Location
Geographic location: Chicago
Physical machine name: CHISVR1
Path code
Prototype: PY900
Production: PD900
LAN connections
15 production Workstations
LAN connections
15 production Workstations
15 CRP Workstations

This table describes the assumptions used by the Tier 1 Denver location in this case study:

Characteristic	Setting
Enterprise Server name	HOME1
Deployment Server name	Denver: DENSVR1 Atlanta: ATLSVR1 Chicago: CHISVR1
Prototype workstations	Denver: 15 Atlanta: 0 Chicago: 15

Characteristic	Setting
Production workstations	Denver: 0 Atlanta: 15 Chicago: 15
JD Edwards EnterpriseOne release	E920
Deployment tier	Denver: 1 Atlanta: 2 Chicago: 2
Path codes	Denver: PD920, PY920 Atlanta: PD920 Chicago: PD920, PY920

Multitier Deployment Configuration Steps for the Case Study

These steps summarize the steps necessary to configure the system for multitier deployment:

1. Define the deployment locations.

   Define the deployment locations on the deployment server (DENSVR1 in this example). Use Oracle's JD Edwards EnterpriseOne Deployment Locations Application program (P9654A) to define all deployment locations.

   For this case study, complete these fields to define three locations, one for each deployment location in Denver, Atlanta, and Chicago:

Field	Value
Location	Enter the name of the deployment location. In this case study, you assign these names for each physical deployment location: **Denver, Atlanta,** and **Chicago.**
Description	Enter a description (any value up to 30 characters) for each deployment location; for example: Denver: **Denver - Tier 1** Atlanta: **Atlanta - Tier 2** Chicago: **Chicago - Tier 2**
Location Code	Enter the current location for deployment; for example, **DEN.**

Field	Value
Parent Location	Enter the name of the parent location for the location that you are adding; for example, **Corporate.**

2. Create deployment server definitions.

   Use the JD Edwards EnterpriseOne Deployment Locations Application program (P9654A) to create a definition for each deployment server at the deployment locations that you created. For this case study, you need to define a deployment server for Atlanta and Chicago. The deployment server in Denver is already defined because it is the primary (tier 1) server.

   For this case study, complete the fields on the Deployment Server Revisions form as described in this table:

Field	Value
Machine Name	Enter the name of the physical machine. In this case study, enter these values: Denver: **DENSVR1** Atlanta: ATLSVR1 Chicago: CHISVR1
Description	Enter a description (any value up to 30 characters). For example, **Multitier Deployment - Denver.**
Release	Enter the number of the release. For example, E920.
Primary User	Enter the primary user for the machine that you entered.
Server Share Path	Enter the name of the shared directory for the path code in which system files and other files reside. For example, \E920.

3. Schedule the package.

   Schedule the package to be deployed from the tier 1 deployment server to the tier 2 deployment servers through the JD Edwards EnterpriseOne Deployment Director program (P9631) or Multi Tier Deployment batch program (R98825C).

Note:
- *Defining Deployment Parameters*.

Defining Deployment Servers

This section provides an overview of defining deployment servers, lists prerequisites, and discusses how to:

- Define a new deployment server.
- Revise an existing deployment server.

Understanding Defining Deployment Servers

The JD Edwards EnterpriseOne Deployment Locations Application program (P9654A) enables you to either add a new deployment server definition or modify an existing definition. When you add a new deployment server definition, the system creates a record in the F9654 table for each deployment location. Each server at each deployment location must be defined.

Typically, the table contains one record for the primary deployment server and one record per deployment location for each release. If you have multiple releases, you must create multiple records for the servers at each deployment location.

If you used Oracle's JD Edwards EnterpriseOne Remote Location Workbench to create deployment server definitions when you installed the software, you do not need to define deployment servers again.

In many situations, you might need to modify the definition for a deployment server that you already defined. For example, you would need to change the definition if the server share path or release changes, or if you want to designate a different server as the primary deployment server. The process for revising an existing deployment server definition is similar to the process for adding a new definition.

For more information on the Installation Workbench, see the appropriate JD Edwards EnterpriseOne Applications installation guide.

Prerequisites

Before you complete the tasks in this section:

- Ensure that you understand the CNC concepts that enable you to define and implement packages, workstation installations, path codes, central objects, replicated objects, and user profiles.

- Ensure that adequate disk space exists on the disk drive for the machine that you will be using as a tier deployment location.

  Each full package that you deploy adds approximately 1.4 GB. The amount of required disk space varies, depending on the amount of data that you replicate to a Supported Local Database.
- If you are adding a new definition for a deployment server at a remote location, you first need to define that location.

  > **Note:** It is recommended that you do not define deployment locations with a DEV path code. The DEV path code is normally associated with developers who are not located at remote locations.

  See *Defining Locations*.

Form Used to Define a Deployment Server

Form Name	FormID	Navigation	Usage
Deployment Server Revisions	W9654AC	Package and Deployment Tools (GH9083), Machine Identification Find the location where the deployment server resides and expand the tree, if necessary, to display the different machine types. Select the Deployment Server heading and then click Add, or find the existing deployment server you want to modify and click Select.	Define a new deployment server or revise an existing deployment server for multitier deployment.

Defining a New Deployment Server

Access the Deployment Server Revisions form.

1. Enter the name of the machine on the network in the Machine Name field.

   Because you are entering a definition for a server other than the primary deployment server, the **Primary Deployment Server** field is not available. You can enter this field only if you first delete the definition for the current primary deployment server.
2. Enter the release number as defined in the release master in the Release field.
3. Enter the primary user for the listed machine in the Primary User field.
4. Enter the shared directory for the path code in the Server Share Path field.

   The objects that are stored on a file server will be found in this path.

5. Select Path Code from the Form menu.
6. On the Machine Path Code Revisions form, enter the path code for the primary or base location from which the secondary location will receive the package.

   When you enter the path code, the server share path will use the base server share path as the default for that machine.

   > **Note:** You must specify the path code for each deployment server at all secondary locations. If you do not indicate the path code, multitier deployment may not work.

7. If the package includes a user-defined foundation or data, specify those items by selecting Foundation or Data from the Form menu.
8. On the Deployment Server Revisions form, click OK.

Revising an Existing Deployment Server

Access the Deployment Server Revisions form.

1. Modify any of these fields:
   - Description
   - Release
   - Primary User
   - Server Share Path
   - Primary Deployment Server

     If you are modifying the primary deployment server, you can also change the primary deployment server. For any other server, you cannot change the value in this field. Enter **1** to indicate that the deployment server is the primary deployment server. You can have only one primary deployment server.
2. Select Path Code from the Form menu.
3. On the Machine Path Code Revisions form, enter the path code for the primary or base location from which the secondary location receives the package.

   When you enter the path code, the system displays the default server share path, which is the base server share path for that machine.

   > **Note:** You must specify the path code for each deployment server at all secondary locations. If you do not indicate the path code, multitier deployment might not work.

4. If the package includes a user-defined foundation or data, specify those items by selecting Foundation or Data from the Form menu.
5. On the Deployment Server Revisions form, click OK.

> **Note:**
> - *Defining Deployment Parameters*.

Distributing Software to Deployment Locations

This section provides an overview of the multitier software distribution process and discusses how to:

- Distribute software through package deployment.
- Schedule packages for multitier deployment.
- Distribute software through the multitier deployment batch process.
- Copy workstation installation programs to deployment locations.

Understanding the Multitier Software Distribution Process

You use the JD Edwards EnterpriseOne Deployment Director program (P9631) or Multi Tier Deployment batch program (R98825C) to distribute software to deployment locations and schedule them for deployment. Use the JD Edwards EnterpriseOne Deployment Director program to define the scheduling parameters or to deploy the package immediately. Otherwise, use a version of Oracle's JD Edwards EnterpriseOne Multi Tier Deployment batch application to distribute the software to deployment locations.

Whether you push or pull the software depends on the machine on which you run the deployment function or batch program for the scheduling program. You push the software if you run the program or report on the primary deployment server or from a workstation. Conversely, you pull the software if you run the program from a workstation at the tier deployment location. In either case, execute the application on the destination deployment location.

Note: If you push the software, you must have full read and write privileges for both deployment servers (that is, the primary deployment server and the tier deployment location or destination machine), even if you do not run the batch application. If you do not have read and write authority on both servers, the deployment will fail.

After the package software is distributed through the JD Edwards EnterpriseOne Deployment Director program or Multi Tier Deployment batch program, you must manually copy the workstation installation programs from the primary deployment server to the tier deployment location. These programs are located in the client portion of the base installation directory.

Form Used to Distribute Software to Deployment Locations

Form Name	**FormID**	**Navigation**	**Usage**
Work with Package Deployment	W9631J	System Administration Tools, Package and Deployment Tools, Package Deployment	Distribute software that was previously defined and scheduled through the Deployment Director program (P9631).

Distributing Software Through Package Deployment

Use this method when you want to distribute software immediately after you define a deployment schedule. If you select this option, the software is distributed immediately, regardless of the timing parameters that you specify in the scheduling fields.

Access the Work with Package Deployment form.

1. Select the Machines option, and then click Find.
2. Select the deployment server to which you want to deploy.

   If you have scheduled to deploy packages to more than one deployment server, you can select the Deployment Server folder to deploy to all applicable servers rather than deploying to each individual server.
3. Select the deployment server name to deploy all packages that are listed under the server name.

   Alternatively, select only the package that you want to deploy.
4. From the Row menu, select Deploy.

   This option launches Oracle's JD Edwards EnterpriseOne Multi Tier Deployment batch program (R98825C), which enables you to deploy to deployment servers.

   You can also use this option to launch a batch application that deploys enterprise server packages. Therefore, if you select an enterprise server name or the Enterprise Server folder on the Work With Package Deployment form, Oracle's JD Edwards EnterpriseOne Enterprise Server Deployment batch program (R98825D)—not the Multi Tier Deployment batch program—launches. When you select a workstation, the Deploy option is not available.

Scheduling Packages for Multitier Deployment

After you create the server package that you want to deploy using multitier deployment, you can use the JD Edwards EnterpriseOne Deployment Director program (P9631) to schedule that package to a server at the same location or a remote location. Verify that the server package is compressed because, unlike client packages, you cannot deploy the server package using multitier deployment unless it is compressed.

Also, before you can deploy a server package with server multitier deployment, the package status must be either 50 (Build Completed Successfully) or 70 (Build Completed with Errors). If the package does not have a status of 50 or 70, it will be unavailable when you schedule the deployment.

See *Scheduling a Package for Deployment*.

Note: When you create a distribution schedule, remember that you cannot schedule multiple packages to deploy at the same time.

Distributing Software Through the Multitier Deployment Batch Process

Access the Work with Batch Versions - Available Versions form.

1. Enter **R98825C** in the Batch Application field and click Find.
2. Select the Multi Tier Deployment version that you want to use.
3. On Version Prompting, click Submit.
4. On Report Output Destination, select one of these options and click OK:
    - On Screen
    - Printer

Copying Workstation Installation Programs to Deployment Locations

These steps ensure that the system runs the JD Edwards EnterpriseOne Client Workstation Installation program locally at the deployment location. If you do not complete these steps, the system searches the base location for the JD Edwards EnterpriseOne Client Workstation Installation program and its associated files, and copies the files across the WAN to the deployment location.

1. Connect to each deployment location and use Microsoft Windows Explorer to drag this client directory to the tier 2 workstation:
    ```
    \\Tier1DeploymentServerName\E920\OneWorld Client Install
    ```
2. Open the package.inf file and locate the [FileLocations] section.

    Change the PackageInfs line to reflect the machine name of the deployment server and the environment at the deployment location; for example:
    ```
    PackageInfs=\\machine name\environment\ENVIRON\Evapps\appl_pgf\package_inf
    ```
3. Save your changes by selecting Save from the File menu.
4. Select Exit from the File menu.

In the case of multitier deployment, the client installation program resides on the tier deployment location in the client subdirectory that is subordinate to the base installation directory. The workstation that is attached to the deployment location must have read access to this directory on the deployment location to install the workstation package.

For more information on installing workstations, see the appropriate JD Edwards EnterpriseOne Application installation guide.

Deploying Server Packages in a Multitier Network

This section provides an overview of multitier deployment of server packages and discusses how to schedule a server package for multitier deployment.

Understanding Multitier Deployment of Server Packages

Server multitier deployment lets you automatically deploy a server package from one deployment server to another. The target deployment server to which you are deploying the package can be either in the same location as the source deployment server that sends the package or in one or more remote locations.

You typically build packages on the primary deployment server at the base location, but using a different server for installations from the base location might have advantages. In some situations, you might prefer to deploy a server package to a server at a remote location rather than require remote users to access the server package over a WAN.

Server package multitier deployment enables users to select which package builds they want to deploy. For example, if you are building a Prod package for multiple server platforms, you can select the build for the platform that has been successfully built. The benefit of having the ability to select builds is that users are no longer required to wait to install a new client package.

You use the JD Edwards EnterpriseOne Deployment Director program (P98631) for server multitier deployment. When you deploy a server package, the system copies these components:

- Foundation and data.
- Subdirectories of the package name directory:
  - bin32
  - include
  - lib32
  - obj
  - source
  - spec
- Subdirectories of the server build directories:
  - bin32
  - spec
  - obj
  - lib32
- ServerPackage.inf file in the server build directories:
  - The package.inf file.
  - The pack directory that is subordinate to the package name directory.

When server-only builds are deployed without client builds, only the pack subdirectory is copied.

This table describes where major package components are copied from and to during the deployment process; the server share path is derived from the F9651 table:

Package Component	Copied From	Copied To
Package	The path indicated in the SrcDirs section of the package.inf file.	<server share path>\<path code>\<package name> on the destination deployment server.

Package Component	Copied From	Copied To
Foundation and data	The path indicated in the SrcDirs section of the package.inf file.	<server share path>\systemcomp on the destination deployment server if the default location is selected.
Package.inf file	The path indicated in the F00942 table if the source deployment server is the primary deployment server in the base location. Otherwise, the path to the package.inf file in the F9651 table.	<server share path>\package_inf on the destination deployment server.

Smart Deployment

Server multitier deployment incorporates the smart deployment feature, which makes available only the server packages that match the available servers for the package destination. For example, if server packages exist at the base location for the Solaris and the *IBM i* but the destination location has only a Solaris, only the Solaris package is made available for deployment to that location. You cannot deploy a server package unless the package destination supports that server platform.

Even if you have multiple locations, smart deployment ensures that only the server packages that match the destination platform are available.

Automatic Package.inf File Updating

When you deploy a server package to a remote location, these sections of the package.inf file are updated:

- SrcDirs
- Attributes
- DeploymentServerName
- Location

The package.inf file is not copied when you deploy to a deployment server in the base location.

Prerequisite

Assemble and build the server package.

Form Used to Schedule a Server Package for Multitier Deployment

Form Name	FormID	Navigation	Usage
Package Deployment Targets	W9631B	Package and Deployment Tools (GH9083), Package Deployment Click Add to launch the Deployment Director, and then click Next.	Select a package to deploy.

Scheduling a Server Package for Multitier Deployment

Access the Package Deployment Targets form.

1. Select the Deployment Server option.
2. On the Package Deployment Attributes form, enter the deployment date and time, and select deployment options.
3. On the Deployment Server Selection form, enter the machine name to which you want to deploy the server package.

   Select the machine by double-clicking the row header.
4. On the Build Selection form, select the build (or version) of the package that you want to deploy by double-clicking the row header.

   Because of the Smart Deployment feature, the system enables you to select only the package builds that match the configuration at the destination location. For example, if package builds exist for an *IBM i* and a Solaris but the destination location has only a Solaris, you cannot select the *IBM i* build.
5. Click Close to exit the Build Selection form.
6. On the Work With Package Deployment form, click End to finish the server package scheduling process.

Note:

- *Distributing Software to Deployment Locations*.

12 Appendix A - Adding a Security Override for Package Build

Understanding Security Overrides for Package Build

Before you build a package, you must have a security administrator add a security override if your system meets both of these conditions:

- The database that you are using is Oracle, SQL, or *DB2 for Linux, UNIX, and Windows database* .
- The security server is enabled.

A security override is required so that the package build process can create the metadata repository tables in central objects. To add a security override, a security administrator must first add a system user for the owner of the central objects data source, and then add a security override to the user who will be building packages.

Adding a System User for Each Data Source

Starting with Tools Release 9.2.5.0 you can use long database passwords. You must perform the software update process with the Security Server turned on if you are using long database passwords. The system user passwords must be changed using P98OWSEC or P98LPSEC and you must add security overrides to all data sources prior to applying ESUs with the Security Server turned on.

Add security overrides for the following:

• System – 920

• Data Dictionary - 920

• Object Librarian – 920

• Data Dictionary – PS920 for PS920

• Business Data, Control Tables, Central Objects and Versions for each environment

For example Business Data – TEST, Control Tables – Test, Central Objects – Test, Versions – DV920

Note:

You cannot specify a password longer than 10 characters if using the [DSPWD] section in the jde.ini.

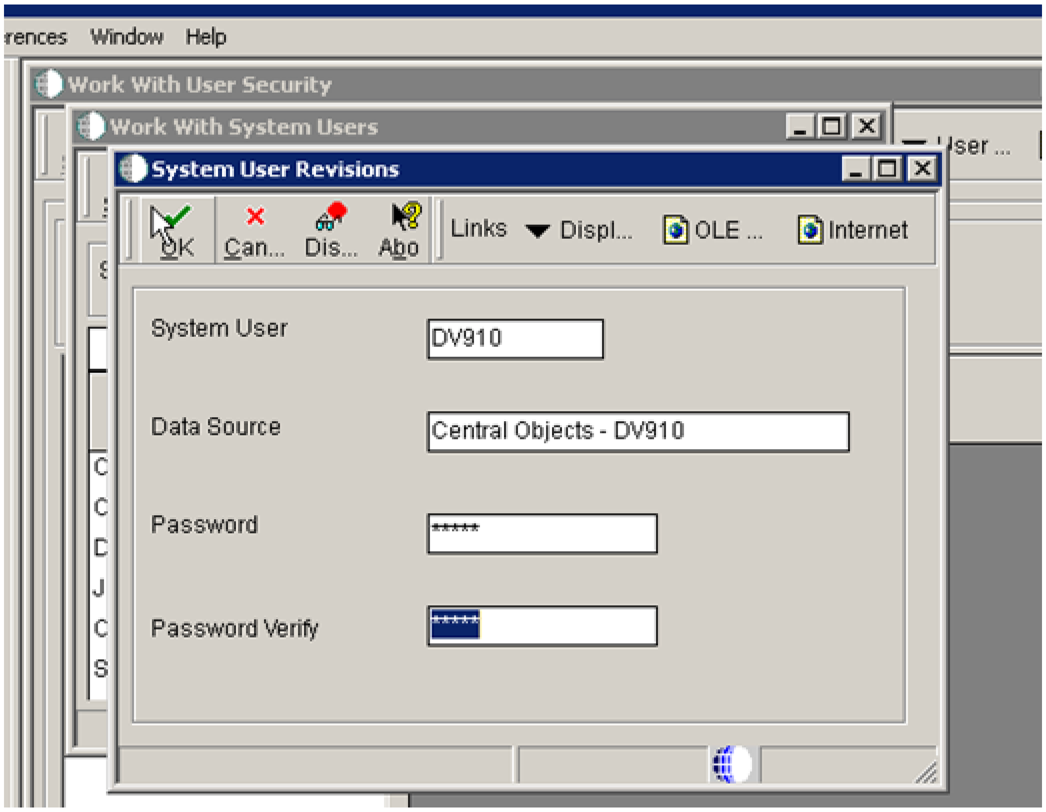
To add a system user for each data source:

Log into a web development client workstation or into the *DEP environment on the deployment server.

1. Enter **P98OWSEC** in the Fast Path to access System User Revisions. If using long database passwords enter **P98LPSEC** in the Fast Path.
2. On Work with User Security, from the Form menu, select Add System User.

3. On Work with System Users, in the System User field, enter the appropriate data source owner, for example **DV920 .**
4. Click the Find button.
5. If no values are returned, add the data source owner as a system user:

   a. Click the Add button.
   b. On System User Revisions, complete these fields: System User (for example, DV920), Data Source (for example, Central Objects - xxx), Password (for example, DV920 or whatever the database user's password is), and Password Verify.



System User: DV910
Data Source: Central Objects - DV910
Password: *****
Password Verify: *****

   c. Click OK.
6. Repeat these steps for each of your defined data sources, such as Central Objects, Data Dictionary, Object Librarian, and System.
7. When you are finished adding records, click the Cancel button and the Close button to return to the Work with User Security form.

Adding a Security Override for the User Building Packages

Starting with Tools Release 9.2.5.0 you can use long database passwords. You must perform the software update process with the Security Server turned on if you are using long database passwords. The system user passwords must be changed using P98OWSEC or P98LPSEC and you must add security overrides to all data sources prior to applying ESUs with the Security Server turned on.

Add security overrides for the following:

• System – 920

• Object Librarian – 920

• Data Dictionary – 920

• Business Data, Control Tables and Versions for each environment

For example Business Data – TEST, Control Tables – Test, Versions – DV920

Note: You cannot specify a password longer than 10 characters if using the [DSPWD] section in the jde.ini.

To add a security override for the user who will be building packages:

1. On Work with User Security, enter the User ID for the previously defined data source, such as **JDE**, and then click the Find button.
2. From the Form menu, select Add Data Source.
3. On Add Data Source, complete these fields:
    - User ID
    - Data Source
    - System User
4. Follow the above steps to add security overrides for each data source, such as Central Objects - xxx, System - xxx, and Data Dictionary - xxx.
5. When you are finished, sign out of JD Edwards EnterpriseOne and restart the JD Edwards EnterpriseOne server so these necessary changes will take effect.

13 Appendix B - Running-64-bit Code Converter (Release 9.2.3)

Running the 64-bit Code Converter (Release 9.2.3)

You can use this UBE to convert a particular path code's include and source files to 64-bit. The path code must already have the 32-bit include and source in the repository table (F98780R). This conversion will either happen the first time package build is run with a 64-bit client foundation or by using this UBE (R9664CONV) to convert the path code. Using R9664CONV will save time when running a package build for the first time with the 64-bit foundation.

To run the 64-bit code converter:

1. Go to Batch Versions to run the UBE.
2. Select **Row**, then **Processing Options.**
3. Specify values for these parameters:

    - **Process Mode**

      0 = Path code. Include and source files in the repository (F98780R) table are converted to include64 and source64 files.

      1 = Local. Local include and source directories are converted to include64 and source64 directories.
    - **Path Code**

      Specify the path code to convert.
    - **Ignore 64Bit Pathcode Flag**

      Blank = Do not ignore.

      '1' = Ignore.

      When the process finishes, it sets a field in the F00942T table to '1' to indicate that this path code has been converted. If you need to rerun this UBE, then use '1' to ignore this setting in the field. Use this processing option to override an existing setting.

14 Appendix C - 64-bit Repository Insert Logs (Release 9.2.3)

Overview

During the insert into the repository, logs are produced under the *<package name>*/work directory.The following three processes perform inserts into the database and produce logs:

- **Insert of 32-bit Files.** If running a foundation that is 32-bit or your path code is only 32-bit, the process retrieves the include, source, and res files from the Deployment Server path code and inserts the include, source, and res files into the database.
- **Insert of 64-bit Files**. If after process one, the package is built with a foundation that is 64-bit, the process retrieves the record from the F9870R, converts the include and source into include64 and source64, and inserts all four back into the repository.
- **Insert of 32-bit and 64-bit Files.** If process one has never been performed and the foundation is 64-bit, the process retrieves the include, source, and res files from the Deployment Server path code, converts the include and source to include64 and source64, and then inserts all four files into the database. For applications, it inserts the res bitmaps associated with that application and there is no conversion with the bitmaps.

Insert of 32-bit Files

This process queries the F9860 table for all objects. It checks each object to see if there is a file associated with it on the Deployment Server under the path code that it is converting. The include, source, and res is then inserted into the F98780R and F98780H tables along with the central object specs.

The objects that have files associated with them are business functions, tables, business views, UBEs, and applications. All business functions and tables have a file. Business functions that are NER (N*.c) and tables with TER(f*.c) have a file, but in Tools Release 9.2.3.0, the files are not loaded into the repository. There is no need to store these in the repository because these are generated during the package build process and during the OMW process. If a business view has a .h file, the file is named B<*view name*>.h. If a UBE has a .h file, the file is named <*ube name*>.h. Applications might have a bitmap associated with them and it would be located in the res directory.

The process produces these logs:

- APPLog.log
- UBELog.log
- BSFNLog.log
- BSVWLog.log
- TBLELog.log
- GTLog.log
- DSTRLog.log

- NotInsertLog.log
- MisMatchLog.log
- NotCheckinLog.log
- 88ObjLog.log

The first seven logs list all objects that were selected from the F9860 table and processed. Inclusion in these logs does not mean that there was a file associated with an object, just that it was selected from the F9860 table.

For example:

```
Fri Mar 02 00:16:48  -            0 - BSFN Name B0100084
Fri Mar 02 00:16:52  -            1 - BSFN Name B0101500

Fri Mar 02 00:16:57  -           54 - APPL Name P43250
Fri Mar 02 00:17:36  -          503 - APPL Name P4371
```

where it is formatted:

<date and time> - <sequence> <object type> Name *<name of object>*.

The *<sequence>* is the sequence of when it was retrieved from the database.

The remaining logs and their contents are:

- **NotInsertLog.log.** Lists the objects that were not inserted into the database because there were no files associated with the objects.
- **NotCheckinLog.log.** Lists all objects that were not checked into the path code.
- **88ObjLog.log.** Lists all objects that were at a system code 88 or Not Applicable.
- **MisMatchLog.log.** Lists all the objects that might have a file but when the process tried to insert it into the database, the file was not found. Many of these will be NER, which does not matter because they are not needed.

Insert of 64-bit Files

This process queries the F9860 table for all objects that have a file associated with them. The query uses the condition where the F9860->sigtffu1= 1 and not equal to "GT", "APP", "DSTR" and "88"(Not Applicable).

Note: This process is performed after step one has already been processed.

For each object that is retrieved from the F98780R, it will first check if the object has already been converted (through the ESU process or OMW) and if there is no include64 and source64 found, it will convert the include and source files into an include64 and source64. It will then insert all four files into the F98780R and F98780H along with the central object specs.

The objects that have files associated with them are business functions, tables, business views, and UBEs. All business functions and tables will have a file. Business functions that are NER (N*.c) and tables with TER(f*.c) have a file, but in 9.2.3.0, the files are not loaded into the repository. There is no need to store these in the repository because these are generated during the package build process and during the OMW process. If a business view has a .h file, the file is named B*<view name>*.h. If a UBE has a .h file, the file is named *<ube name>*.h. All these files are converted into an include64 and source64 directory.

The process produces these logs:

- UBELog_64.log
- BSFNLog_64.log
- BSVWLog_64.log
- TBLELog_64.log
- NotCheckinLog_64.log
- NotConvertLog_64.log

The first four logs list all objects that were selected from the F9860 table and processed.

For example:

```
Fri Mar 02 00:16:48  -            0 - BSFN Name B0100084
Fri Mar 02 00:16:52  -            1 - BSFN Name B0101500
```

where it is formatted:

<date> - *<sequence>* *<object type>* Name *<name of object>*.

The *<sequence>* is the sequence of when it was retrieved from the database.

The NotConvertLog_64.log lists all objects that were not converted. This could mean it found an include64 and source64 file so it did not have to convert them.

The NotCheckinLog_64.log lists all objects that were not checked into the path code.

Insert of 32-bit and 64-bit Files

This process queries the F9860 for all objects.

Note: This process is used if process one was never performed and the client foundation is 64-bit.

The query checks each object to see if there is a file associated with it on the Deployment Server, under the path code it is converting. If it is a .c or .h file, it will then convert the file into the include64 and source64 files. The include, source, include64, and source64 or res files are then inserted into the F98780R and F98780H tables along with the central object specs.

The objects that have files associated with them are business functions, tables, business views, UBEs and applications. All business functions and tables will have a file. Business functions that are NER (N*.c) and tables with TER(f*.c) tables will have a file but in 9.2.3.0, the files are not loaded into the repository. There is no need to store these in the repository because these are generated during the package build process and during the OMW process. If a business view has a .h file, the file is named B<*view name*>.h. If a UBE has a .h file, the file is named <*ube name*>.h. Applications might have a bitmap associated with them, which would be located in the res directory.

The process produces these logs:

- APPLog.log
- UBELog.log

- BSFNLog.log
- BSVWLog.log
- TBLELog.log
- GTLog.log
- DSTRLog.log
- NotInsertLog.log
- MisMatchLog.log
- NotCheckinLog.log
- 88ObjLog.log

The first seven logs list all objects that were selected from the F9860 table and processed. Inclusion in these logs does not mean that there was a file associated with an object, just that it was selected from the F9860 table.

For example:

```
Fri Mar 02 00:16:48  -           0 - BSFN Name B0100084
Fri Mar 02 00:16:52  -           1 - BSFN Name B0101500
Fri Mar 02 00:16:57  -          54 - APPL Name P43250
Fri Mar 02 00:17:36  -         503 - APPL Name P4371
```

where it is formatted:

<date> - *<sequence> <object type>* Name *<name of object>.*

The *<sequence>* is the sequence of when it was retrieved from the database.

The remaining logs and their contents are:

- **NotInsertLog.log.** Lists the objects that were not inserted into the database. This means there was not a file associated with an object so it was not inserted.
- **NotCheckinLog.log.** Lists all objects that were not checked into the path code.
- **88ObjLog.log.** Lists all objects that were at a system code 88 or Not Applicable.
- **MisMatchLog.log.** Lists all the objects that might have a file but when the process tried to insert it into the database, the file was not found. Many of these will be NER, which does not matter because they are not needed.

15 Glossary

Build Configuration File

Configurable settings in a text file that are used by a build program to generate ANT scripts. ANT is a software tool used for automating build processes. These scripts build published business services.

check-in repository

A repository for developers to check in and check out business service artifacts. There are multiple check-in repositories. Each can be used for a different purpose (for example, development, production, testing, and so on).

control tables merge

A process that blends a customer's modifications to the control tables with the data that accompanies a new release.

embedded event rule

An event rule that is specific to a particular table or application. Examples include form-to-form calls, hiding a field based on a processing option value, and calling a business function. Contrast with the business function event rule.

Location Workbench

An application that, during the Installation Workbench process, copies all locations that are defined in the installation plan from the Location Master table in the Planner data source to the system data source.

Object Librarian Merge

A process that blends any modifications to the Object Librarian in a previous release into the Object Librarian in a new release.

Package Workbench

An application that, during the Installation Workbench process, transfers the package information tables from the Planner data source to the system-release number data source. It also updates the Package Plan detail record to reflect completion.

serialize

The process of converting an object or data into a format for storage or transmission across a network connection link with the ability to reconstruct the original data or objects when needed.

Specification merge

A merge that comprises three merges: Object Librarian merge, Versions List merge, and Central Objects merge. The merges blend customer modifications with data that accompanies a new release.

Versions List merge

The Versions List merge preserves any non-XJDE and non-ZJDE version specifications for objects that are valid in the new release, as well as their processing options data.

vocabulary override

An alternate description for a data dictionary item that appears on a specific JD Edwards EnterpriseOne form or report.

workgroup server

A server that usually contains subsets of data replicated from a master network server. A workgroup server does not perform application or batch processing.

Index