JD Edwards EnterpriseOne Tools

Development Tools: Report Printing Administration Technologies Guide

Release 8.98 Update 4

E14709-02

March 2011

ORACLE

JD Edwards EnterpriseOne Tools Development Tools: Report Printing Administration Technologies Guide, Release 8.98 Update 4

E14709-02

Contents

5 Working with Report Printing Administration

Preface

Welcome to the JD Edwards EnterpriseOne Tools Development Tools: Report Printing Administration Technologies Guide.

Audience

This guide is intended for reporting and analytics administrators who are responsible for managing the different output options available for viewing a report.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/us/corporate/accessibility/index.html.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/support/contact.html or visit http://www.oracle.com/accessibility/support.html if you are hearing impaired.

Related Documents

You can access related documents from the JD Edwards EnterpriseOne Release Documentation Overview pages on My Oracle Support. Access the main documentation overview page by searching for the document ID, which is 876932.1, or by using this link:

https://support.oracle.com/CSP/main/article?cmd=show&type=NOT&id=876932.1

To navigate to this page from the My Oracle Support home page, click the Knowledge tab, and then click the Tools and Training menu, JD Edwards EnterpriseOne, Welcome Center, Release Information Overview.

This guide contains references to server configuration settings that JD Edwards EnterpriseOne stores in configuration files (such as jde.ini, jas.ini, jdbj.ini, jdelog.properties, and so on). Beginning with the JD Edwards EnterpriseOne Tools Release 8.97, it is highly recommended that you only access and manage these settings for the supported server types using the Server Manager program. See the Server Manager Guide on My Oracle Support.

Conventions

The following text conventions are used in this document:

Convention	Meaning
Bold	Indicates field values.
Italics	Indicates emphasis and JD Edwards EnterpriseOne or other book-length publication titles.
`Monospace`	Indicates a JD Edwards EnterpriseOne program, other code example, or URL.

1

Introduction to JD Edwards EnterpriseOne Report Printing Administration Technologies

This chapter contains the following topics:

- Section 1.1, "JD Edwards EnterpriseOne Report Printing Administration Technologies Overview"
- Section 1.2, "JD Edwards EnterpriseOne Report Printing Administration Technologies Implementation"

1.1 JD Edwards EnterpriseOne Report Printing Administration Technologies Overview

JD Edwards EnterpriseOne Report Printing Administration Technologies addresses the printing properties available in JD Edwards EnterpriseOne Report Design Aid, the printing properties presented at runtime, how to define printers for reporting, and the different output options available for JD Edwards EnterpriseOne reports.

1.2 JD Edwards EnterpriseOne Report Printing Administration Technologies Implementation

This section provides an overview of the steps that are required to implement JD Edwards EnterpriseOne Report Printing Administration Technologies.

1.2.1 JD Edwards EnterpriseOne Report Printing Administration Technologies Implementation Steps

This section provides an overview of the steps that are required to implement JD Edwards EnterpriseOne Report Printing Administration Technologies.

In the planning phase of the implementation, take advantage of all JD Edwards sources of information, including the installation guides and troubleshooting information.

The following list contains the high-level steps for the JD Edwards EnterpriseOne Report Printing Administration Technologies implementation.

- Set up permissions to access and use JD Edwards EnterpriseOne Object Management Workbench (OMW) and the JD Edwards EnterpriseOne Printers application using JD Edwards EnterpriseOne Security Workbench.

  See "Using Security Workbench, Managing Application Security" in the*JD Edwards EnterpriseOne Tools Security Administration Guide*.

- Add yourself to the system in a developer role so that you have permissions to create and modify JD Edwards EnterpriseOne objects.

  See "Configuring User Roles and Allowed Actions, Setting Up User Roles" in the *JD Edwards EnterpriseOne Tools Object Management Workbench Guide* .

- Set up permissions to create OMW projects.

  See "Configuring User Roles and Allowed Actions, Setting Up User Roles" in the *JD Edwards EnterpriseOne Tools Object Management Workbench Guide* .

- Set up save locations to enable you to save JD Edwards EnterpriseOne objects that are not ready to be checked in.

  See "Configuring Object Save Locations" in the *JD Edwards EnterpriseOne Tools Object Management Workbench Guide*.

2

Understanding Report Printing Administration Technologies

This chapter contains the following topics:

2.1 Report Output

In JD Edwards EnterpriseOne Report Design Aid (RDA), you create reports and define specific printing properties to affect the report output. The report developer can process reports from the Microsoft Windows client. The batch engine can process reports on various servers or on the workstation.

After development is complete, the reports and associated batch versions are checked in and advanced through the development cycle. The system administrator then builds a package and deploys the reports and batch versions to the enterprise server. The reports and batch versions are generated to HTML so that they can be run from the web client.

Report output from the Microsoft Windows client can be in these formats:

- PDF

  View the report online in Adobe Acrobat Reader.

- Comma separated values (CSV)

  Export the report to a CSV spreadsheet application.

- Output stream access (OSA)

  Export the report to a third-party product using OSA interfaces.

- Printed copy

  Send the report to a printer.

- XML

  XML output is generated by running a Report Definition.

  See *JD Edwards EnterpriseOne Tools BI Publisher For JD Edwards EnterpriseOne Guide*.

After report processing is complete, output management handles the generation and output of the report.

From the web client, the batch engine processes reports on various servers. Report output from the web client can be in these formats:

- PDF
- CSV
- Output Stream Access (OSA)
- Printed copy
- XML Publisher

  XML Publisher output is generated by running a Report Definition.

  See *JD Edwards EnterpriseOne Development Tools 8.97 XML Publisher Guide*.

Viewing the report on screen is not an option at runtime from the web client. However, once the report has been processed, you can view the report online using options on the Row menu.

2.2 Output Management

Output management refers to managing the different output options available for viewing a report. You can view reports in different file types, send them to different printers, and create output in different forms or paper sizes. JD Edwards EnterpriseOne accommodates simple output processes such as viewing the PDF of a report online or sending it to a network printer. You can also use more complex processes such as sending versions of a report to different printer drawers or defining versions to print to different printers across the country.

Some output options are defined in initialization files. For the Microsoft Windows client, the jde.ini is read at runtime by the Microsoft Windows client. For the web client, the jas.ini is read at runtime by the JAS server.

3

Defining Print Properties for Reports

This chapter contains the following topics:

3.1 Understanding Print Properties

You add a new printer to the system and define the print properties using the JD Edwards EnterpriseOne Printers (P98616) application. You define printers for use by a specific user role or for all users. You have the option of associating printers with:

- A specific report.
- A specific version of a report.
- All reports.

JD Edwards EnterpriseOne Report Design Aid (RDA) incorporates printing properties to determine the format of the report output. The basic hierarchy of print properties is:

1. Print properties defined at runtime override all other printer definitions.

   This includes definitions set in the JD Edwards EnterpriseOne Printers application, report template, and batch version.

2. Print properties defined in batch versions override the properties defined in the associated report template.
3. Print properties defined in RDA override the properties defined in the JD Edwards EnterpriseOne Printers application.
4. Print properties defined in the JD Edwards EnterpriseOne Printers application are the default properties used for batch applications.

> **Note:** When you modify print properties in an existing report template, the modifications are not reflected in any of the batch versions that exist prior to making the modification.

3.2 Modifying Print Properties

This section provides overviews of designated printers, the Initialize Logical Printer Name system function, paper types, exporting to comma separated values (CSV), and output stream access (OSA) interfaces, lists the prerequisite, and discusses how to:

- Define printers in RDA.

- Select paper types in RDA.
- Export to CSV in RDA.

3.2.1 Understanding Designated Printers

The system administrator defines the default printer to be used with all batch processes. The printer is associated with a user role or with *PUBLIC, a default value that includes all users. The printer is associated with a batch application, a batch version, or *ALL (a default value that includes all batch applications or all batch versions for a specific batch application).

A printer associated with a specific user role overrides a printer associated with *PUBLIC and a specific batch process. You can override these printer definitions by selecting a printer using Print Setup on the File menu in RDA. The printer that you select in RDA is stored in the print specifications, causing the report to always print to the printer that you defined unless overridden at runtime.

When defining a printer in RDA, consider the hierarchy that the system uses to determine the printer that is used at runtime. The system looks first for the printer defined in RDA, if no printer is defined, the system uses the default printer defined in the JD Edwards EnterpriseOne Printers application. The printer hierarchy is:

1. Printer defined at runtime.
2. Printer defined in RDA.

   The printer definition becomes part of the report specifications.
3. Printer defined in the JD Edwards EnterpriseOne Printers application using this hierarchy:
   a. Specific report defined for the user or role.
   b. All reports defined for the user or role
   c. Specific report defined for *PUBLIC.
   d. All reports defined for *PUBLIC.

3.2.2 Understanding the Initialize Logical Printer Name System Function

You can use the Do Initialize Printer event to specify a printer to be used by the system when the batch application processes. The Do Initialize Printer event is a report level event located on the File menu. Using this event, you can print the same report to different printers based on criteria that you define. The event rules located on this event are the first event rules processed at runtime. The event rules are also processed each time a subsystem trigger record is processed. The Initialize Logical Printer Name system function resolves and validates the printer name that you pass to it. The batch engine uses the printer name, if valid, to obtain a printer device context. Portions of this device context can be overridden when the appropriate settings in the report specifications are set.

The Initialize Logical Printer Name system function is ignored if placed on any event other than the Do Initialize Printer event. If you place this system function on a different event, the system generates a message in the jdedebug log.

3.2.3 Understanding Paper Types

When defining the paper size for reports, you can select from predefined paper sizes or you can enter custom paper dimensions. The standard predefined selections available in the JD Edwards EnterpriseOne Printers application are:

- A4
- Legal
- Letter

You must define one of these selections as the default paper type. You can override the default paper type from Print Setup in RDA.

The paper types defined in the JD Edwards EnterpriseOne Printers application are stored in the Paper Definition (F986162) table. RDA inherits the paper size from this table. You must define additional paper types for use in RDA using the JD Edwards EnterpriseOne Printers application.

You can use a selection of different units of measurement to define custom paper sizes in the JD Edwards EnterpriseOne Printers application. The minimum definable *width* in inches is two inches, and the maximum is 21 inches. The minimum definable *height* in inches is two inches and the maximum is 24 inches. In RDA, you can also define custom paper sizes from the Print Setup form. Definitions that you set up in RDA override the definitions set up in the JD Edwards EnterpriseOne Printers application.

3.2.4 Understanding Exporting to CSV

In addition to viewing the report in a CSV file, you can manipulate the report data after the report finishes processing. To view report data in a spreadsheet program, such as Microsoft Excel or Lotus 123, select the export to a CSV option. You can select the CSV option using these methods, each with a different result:

- In RDA for the report template.

  Use this option to ensure that the report is output to a CSV file every time *any* of the associated batch version are run.

  Select a report template and in RDA, select the Export to CSV option in Print Setup.

  The Export to CSV option is selected by the system at runtime. When a report template is defined to export to CSV for every instance, you can clear the Export to CSV option at runtime if you do not want the batch version to export to a CSV file for a single submission.

- In RDA for the batch version.

  Use this option to ensure that the report is output to a CSV file every time *this* specific batch version is run.

  Select a batch version and in RDA, select the Export to CSV option in Print Setup. The batch version specifications will include information to export the output to a CSV file.

  The Export to CSV option is selected by the system at runtime. When a batch version is defined to export to CSV for every instance, you can clear the Export to CSV option at runtime if you do not want the batch version to export to a CSV file for a single submission.

- At runtime.

  Use this option to output batch versions to a CSV file for a *single* submission only.

When running batch versions locally, select the Export to CSV option to submit the batch version.

When running batch versions on the server, select Export to CSV (Comma Delimited) on the Document Setup tab of the Printer Selection form.

Before exporting report data to CSV, you should review the report and follow these recommendations:

Recommendation	Description
Set the horizontal grid alignment to 52 and select the snap to grid option.	The default column width in spreadsheet programs is equivalent to 52 units in RDA. For best results, use these grid guideline so that each column included in the report template is equal to a column in the spreadsheet program.
Ensure that no fields of the report overlap.	If a data field overlaps into the next column, the data in the spreadsheet program displays in discrete columns. You can wrap the text in a cell once the data is exported to the spreadsheet program. Delete unused columns in the spreadsheet program and reformat information as needed.
Align data fields vertically.	If data fields are not aligned vertically, they display in separate rows in the spreadsheet program. If more than one data field with the same vertical and horizontal alignment displays in a column, only one of these fields displays in the CSV file. The first field output during the export process occupies the cell in the spreadsheet program.
Format dates properly.	Spreadsheet programs typically use the same date format used in the report.
Use the Auto Format feature.	After the report is exported cleanly, use the Auto Format feature in the spreadsheet program to further format the report.
Countries that use a comma as a decimal marker.	In these countries, the decimal separator is recognized as a comma when the report exports. Tabs are stripped out instead of commas and a tab-separated file with a .txt extension is created. The information transfers as flat text, so totaling columns display only text. You must then set up totaling in the spreadsheet program.

When you export batch versions to CSV:

- A CSV file is created in the PrintQueue directory.
- A PDF file is created in the PrintQueue directory.
- The CSV file is displayed by a spreadsheet program such as Microsoft Excel or Lotus 123, which launches automatically when you run the batch version locally.

  When you run the batch version on a server, select View CSV from the Submitted Job Search form to launch the spreadsheet application and view the file. Only single spacing and portrait orientation is supported for CSV files. Drill-down links are ignored in CSV generation.

3.2.5 Understanding OSA Interfaces

You can select to output reports to third-party software programs using OSA. OSA interfaces enable the third-party program to process and format the data concurrently. The OSA interface must be predefined; several interfaces might exist for a single program, depending on the section types included in the report and the desired output.

OSA uses its own set of commands from a third-party library.

Benefits of using OSA are:

- Employs the formatting options of the target software program.
- Employs the processing power of the target software program.

3.2.6 Prerequisite

Before defining printers, check out an existing report template.

See "How to Open Existing Reports" in the *JD Edwards EnterpriseOne Tools Development Tools: Report Design Aid Guide*.

See Also:

- "Enhancing Reports Using Basic Functionality", "Working with Report, Section, Field, Column, and Row Properties" in the *JD Edwards EnterpriseOne Tools Development Tools: Report Design Aid Guide*.

3.2.7 Defining Printers in RDA

From EnterpriseOne Life Cycle Tools, select Report Management (GH9111), Report Design Aid.

1. From the File menu, select and open a report template that is checked out.
2. From the File menu, select Print Setup.
3. On the Print Setup form, click the browse button immediately following the Printer Name field.
4. On the Printer Search & Select form, select the printer to use for the report and click Select.

3.2.8 Selecting Paper Types in RDA

In RDA, select Print Setup from the File menu to access the Print Setup form.

1. Select a predefined paper type from the drop-down list in the Size field.
2. If an appropriate paper type is not available, select Custom and indicate the paper width and height.

3.2.9 Exporting to CSV in RDA

In RDA, select Print Setup from the File menu to access the Print Setup form.

1. Under Orientation, select Portrait.
2. Under OneWorld Printer, select Export to CSV and click OK.
3. Verify that no columns or fields in the report overlap.

4. From the Layout menu, select Grid Alignment.
5. On the Alignment Grid form, set the horizontal spacing to **52.**
6. Select the Snap to Grid option and click OK.

   The system applies these settings to the entire report.

4

Understanding Print Properties at Runtime

This chapter contains the following topics:

4.1 Batch Version Submission

You must use a batch version to process a report. You can submit batch versions from the Microsoft Windows client using these two methods:

- Locally

  The client immediately launches the batch engine process. You have the option to view the output on screen or send it to a printer.

- On the server

  The server handles batch processing more efficiently than the workstation. When the batch version is submitted to the server for processing, a message is sent to the server with the defined data selection, data sequencing, and other information necessary to run the report, such as report interconnect values and printer information.

You can submit batch versions to the server using these methods:

- The JD Edwards EnterpriseOne Batch Versions (P98305) application on the Microsoft Windows client.
- Another report or interactive application using a report interconnect.

  You must define the report interconnect in JD Edwards EnterpriseOne Report Design Aid (RDA). The system launches the child batch version from a batch application or interactive application submitted from either the Microsoft Windows client or the web client.

- The JD Edwards EnterpriseOne Submit Job (P98305W) application on the web client.

  On the web client, batch versions are submitted *only* to the server.

- From the JD Edwards EnterpriseOne Menu on the web client .

  For easy access, you can add a report as a task on the JD Edwards EnterpriseOne Menu.

See "Submitting Batch Versions" in the *JD Edwards EnterpriseOne Tools Development Tools: Batch Versions Guide*.

4.2 Jobs Submitted from the Microsoft Windows Client

On the Microsoft Windows client, you can select from these options when submitting batch jobs:

Output Option	Description
On Screen.	Enables you to generate a PDF file that is displayed through the Adobe Acrobat Reader. Adobe Acrobat Reader is launched by the system when the processing of the batch version is complete.
To Printer.	Enables you to modify output options from the Printer Selection form.
Export to CSV.	Enables you to generate both a CSV and a PDF file. The report is displayed through a CSV viewer, such as Microsoft Excel. The CSV viewer is launched automatically by the system when processing of the batch version is complete.
Export using output stream access (OSA).	Enables you to export the report to a third-party software application. The location of the output is determined by the OSA interface. For example, the JD Edwards EnterpriseOne Extended Markup Language (XML) interface creates an XML file in the same location where the PDF and CSV outputs are stored. An OSA library created by the vendor of the third-party software to which you are exporting the report, might store the OSA output in a different location.

See "Submitting Batch Versions," "Submitting Batch Versions from the Microsoft Windows Client" in the *JD Edwards EnterpriseOne Tools Development Tools: Batch Versions Guide*.

Note: For advanced version prompting options that apply to BI Publisher for JD Edwards EnterpriseOne functionality, see *JD Edwards EnterpriseOne Tools BI Publisher For JD Edwards EnterpriseOne Guide*.

4.3 Printer Information

When you launch batch versions from the Microsoft Windows client, the printer information is stored in the pUBEDs structure and passed to the batch engine.

When you submit batch versions to the server, the printer information that is obtained from the Printer Selection form is stored in the Job Control Status Master (F986110) table as a binary large object (BLOB).

When batch jobs are submitted using report interconnects, the printer definition for the child report is inherited from the parent report. The values that are inherited are:

- Printer name
- Print immediate
- SavePDL
- Paper type
- Printer settings
- Number of copies
- Paper source

Export to CSV is not an inherited functionality. If the child report has a printer name defined in its specifications or a custom paper type defined, then these properties override the inherited values.

4.4 The IFS on the iSeries

When you print a batch version on the iSeries server, the resulting report is sent to the IFS in PDF. You can map to the reports using Microsoft Windows Explorer.

The default PrintQueue directory is located under the install directory on the server. If you want to create a custom PrintQueue directory, you need to define the PrintQueue in the initialization file using a valid IFS file name.

This code indicates how the jde.ini should be modified for the IFS on the iSeries:

```
[NETWORK QUEUE SETTINGS]

OutputDirectory = system
```

Where system is the location of the PrintQueue directory.

4.5 Printer Selections at Runtime

When you submit batch versions to a printer, the system looks first for the printer defined in RDA. If no printer is defined in RDA, the system uses the default printer defined in the JD Edwards EnterpriseOne Printers (P98616) application. The system determines a printer based on this hierarchical structure:

1. Printer defined at runtime.
2. Printer defined in RDA.

   The printer definition becomes part of the report specifications.
3. Printer defined in the JD Edwards EnterpriseOne Printers application using this hierarchy:
   a. Specific report defined for the user or role.
   b. Specific report defined for *PUBLIC.
   c. All reports defined for the user or role.
   d. All reports defined for *PUBLIC.

Depending on the printer that is selected, the system selects the corresponding Printer Definition Language on the Advanced tab of the Printer Selection form at runtime.

When batch jobs are submitted using report interconnects, the child report inherits the printer definitions from the parent report. At print time, once the job has completed

processing, you can override the printer for the child report from the JD Edwards EnterpriseOne Work With Servers (P986116) application.

When you submit a job using RUNUBE, if you do not indicate a printer name through the command line parameters, the printer name stored in the specifications is used at print time; otherwise, the default printer is used. At print time, you can override the printer from the JD Edwards EnterpriseOne Work With Servers application when the job has completed processing.

See "Submitting Batch Versions", "Submitting at the Command Line" in the *JD Edwards EnterpriseOne Tools Development Tools: Batch Versions Guide*.

4.6 Paper Type Selections at Runtime

The paper type that you define for the selected printer is used during job submission. From the Print Property tab on the Printer Selection form, you can change the paper type. Depending on the printer that you select, the Paper Type field is automatically populated by the system with the paper type that is defined for that printer.

When you define a custom paper size in RDA, you cannot change the paper type at runtime.

See Also:

- Working with Report Printing Administration.

4.7 Print Orientation Selections at Runtime

The paper orientation that you select in RDA is stored in the report specifications. Orientation from the specifications is displayed on the Print Property tab of the Printer Selection form at runtime. However, you can change the orientation at runtime.

When you define a custom paper size in RDA, the orientation option is unavailable for selection on the Printer Selection form. In the case of line printers, the Characters per Inch (CPI), Characters per Page (CPP), Lines per Inch (LPI), and Lines per Page (LPP) options that you define for the line printer determine the orientation.

4.8 Export to CSV at Runtime

When you select the Export to CSV option in RDA, the definition is stored in the report specifications. This option is displayed in the Report Output Destination form at runtime.

When you submit the batch version to the printer, you have the option of overriding the Export to CSV option on the Document Setup tab of the Printer Selection form. Both a CSV file and a PDF file are created in the PrintQueue directory when the Export to CSV option is selected.

When you submit batch jobs to the server, and the Export to CSV option is defined in RDA, you have the opportunity to modify the option from the JD Edwards EnterpriseOne Work With Servers application.

When batch jobs are submitted using report interconnects, the child report *does not* inherit the export to CSV option from the parent. Therefore, you must select the CSV option in RDA to enable this option for both the child and parent reports.

4.9 Print Settings in the Initialization Files

The Print Immediate option and the Save PDL setting are defined in the initialization files.

4.9.1 The Print Immediate Setting

You can define batch jobs to print immediately by modifying the Print Immediate setting in the initialization file. For the Microsoft Windows client, the Print Immediate setting is located in the jde.ini. For the web client, the Print Immediate setting is located in the jas.ini.

When the Print Immediate setting in the initialization file is set to TRUE, all batch versions print immediately upon processing. When you want specific batch versions to print immediately, use the Print Immediate option on the Version Details form for the individual batch version. The Print Immediate option can be overridden at runtime.

See "Modifying Properties of Batch Versions" in the *JD Edwards EnterpriseOne Tools Development Tools: Batch Versions Guide*.

You can modify the following setting in the jde.ini to specify the Print Immediate functionality for batch jobs:

```
[NETWORK QUEUE SETTINGS]

PrintImmediate=TRUE/FALSE
```

The value of TRUE sends the report output to the printer immediately after processing. FALSE holds the report output in a queue until you select Print from the Row menu on the Submitted Job Search form on the Microsoft Windows client.

You can modify the following setting in the jas.ini to specify the Print Immediate functionality for batch jobs:

```
[OWWEB]

PrintImmediate=TRUE/FALSE
```

The value of TRUE sends the report output to the printer immediately after processing. FALSE holds the report output in a queue until you select Print from the Row menu on the Submit Job - Submitted Job Search form on the web client.

This table describes the results of defining batch jobs to print immediately:

Action	Result
The batch job automatically prints.	You do not have to select the Print menu option on the JD Edwards EnterpriseOne Work With Servers application. By default, at runtime, the Print Immediate option is selected on the Document Setup tab on the Printer Selection form. You can override this setting at runtime.
The child report inherits the definition from the parent report when batch jobs are submitted using report interconnects.	In this case, you cannot override the Print Immediate option at runtime.

You can pass the Print Immediate option as an argument when using RUNUBE from the command line. In this case, the job automatically prints when the PrintImmediate setting in the jde.ini is set to TRUE.

4.9.2 The SavePDL Setting

You can modify the following setting in the jde.ini to specify the SavePDL functionality on the enterprise server:

```
[NETWORK QUEUE SETTINGS]

SavePDL=TRUE/FALSE
```

When you modify the SavePDL setting in the jde.ini to TRUE, both a PDF file and a PDL file are created in the PrintQueue directory. By default, at runtime, the Printer Definition Language File option on the Document Setup tab of the Printer Selection form is selected. However, you can override this option at runtime.

When you modify the SavePDL setting to FALSE, you can select the option at runtime to save the intermediate temporary file. The PDL file resides in the PrintQueue directory. When batch jobs are submitted using report interconnects, the child report inherits the PDL definition from the parent report.

You have the option of modifying the Printer Definition Language File option at runtime. Depending on whether the option is selected or cleared, a PDF and a PDL file are created for that batch job only.

5

Working with Report Printing Administration

This chapter contains the following topics:

5.1 Understanding Report Printing Administration

The JD Edwards EnterpriseOne Printers application provides a single point of entry for configuring printers. The application enables you to define printers for workstations and enterprise servers. These definitions reside in JD Edwards EnterpriseOne tables that are maintained by the JD Edwards EnterpriseOne Printers application.

JD Edwards EnterpriseOne includes a number of predefined reports and batch versions, which you can use to meet the business requirements. If you want to make modifications to one of these predefined reports or batch versions, it is recommended that you copy the report or batch version, and modify the copy. In addition, you can create custom reports using JD Edwards EnterpriseOne Report Design Aid (RDA). The JD Edwards EnterpriseOne batch engine generates these reports in PDF. You can view the PDF files using Adobe Acrobat Reader.

Reports must have at least one batch version before you can process the report. Batch versions do not require user interaction.

When you submit batch versions for processing, you can make some selections at runtime. These selections include:

- Data selection.
- Data sequencing.
- Location where the report processes.
- Logging capabilities to monitor processing.
- The printer to which the report is sent.

See *JD Edwards EnterpriseOne Tools Development Tools: Report Design Aid Guide*.

See *JD Edwards EnterpriseOne Tools Development Tools: Batch Versions Guide*.

5.2 Working with the JD Edwards EnterpriseOne Printers Application

This section provides overviews of the JD Edwards EnterpriseOne Printers application and null pass-through print filters and discusses how to:

- Add printers.
- Define default printers.
- Modify printers.
- Copy printers.
- Delete printers.
- Delete paper types.
- Add null pass-through print filters.
- Search for incorrect printer records.

5.2.1 Understanding the JD Edwards EnterpriseOne Printers Application

JD Edwards EnterpriseOne provides a single application for defining system printers. The JD Edwards EnterpriseOne Printers application uses a director interface with instructions included to guide you through the setup process. From this director, you can:

- Add new printers
- Modify existing printers
- Define default printers

  You can define printers for a combination of users, roles, hosts, environments, and reports.

You can also add and modify the paper types and custom conversion programs that the printers use.

To define a printer for the JD Edwards EnterpriseOne system you must first add a printer. You must complete all of the fields that display on the director forms. Set up printers for each server platform that is used for processing reports in the enterprise. After printers are added, you must define a default printer.

5.2.1.1 Platform Information

When defining the server name and the shared name for printers in the JD Edwards EnterpriseOne Printers application, consider these platform specific guidelines:

- iSeries:

  library name/outqueue name

  For the iSeries, the physical printer name must be the same as the outqueue name. If you use the default QGPL library to store the outqueues, enter only the outqueue name. This information must be entered in upper case.

  Example: `QutputQueue Name`:DEVPRN1

  If the outqueues reside in a library other than the default QGPL library, enter the library name and the outqueue name:

Library Name: QLIBRARY

OutputQueue Name: DEVPRN1

Note: When you qualify the outqueue name with the library name, you avoid possible name conflicts that might result in the submission of the report to an unexpected outqueue.

- Windows NT:

  \\print server name\print shared name

  Examples:

  Print Server Name: corprts1

  Print Shared Name: devprn1

  For Windows NT, enter the name of the print server and the name of the printer. You cannot use spaces or special characters. This information must be entered in lower case. The system uses the print server name along with the print shared name to create the printer name.

- UNIX:

  printer name (no slashes)

  devprn16

  This information must be entered in lower case.

  For printing reports to a non-network printer, leave the printer name field blank.

5.2.1.2 Printer Definition Language (PDL)

When defining the PDL for printers in the JD Edwards EnterpriseOne Printers application, you might be presented with additional options based on the PDL and platform type that you select:

- When defining line printers, you must also define:
  - Characters per inch
  - Columns per page
  - Lines per inch
  - Lines per page

  Note: Use this formula to calculate the paper dimensions:

  Columns per page/Characters per inch = width in inches (85/10 = 8.5)

  Lines per page/Lines per inch = height in inches (66/6 = 11)

- When you define a line printer, the system disables the PostScript and PCL options. The system also disables the detail area at the bottom of the form. Any paper types selected are cleared.
- When you define a line printer that uses the iSeries platform, options appear within a box labeled iSeries Only. Use these fields to define the iSeries encoding that the printer supports:

  - ASCII Encoding
  - EBCDIC Encoding

- When you define a PostScript or PCL printer in combination with the iSeries platform, the ASCII Encoding option is automatically selected and the iSeries Only box is disabled.
- Only users with knowledge of building parameter strings for printers should use the custom option. The custom option uses an advanced feature of the JD Edwards EnterpriseOne Printers application. When you define a custom printer, a field appears beneath the Custom option. Enter the name of the conversion filter that you want to use in this field. You can add or modify conversion filters by selecting Advanced from the Form menu. The Advanced menu option is available only when the Custom option is selected. You can select an existing filter or add new filters on the Work With Conversion Programs form.

  See Understanding Null Pass-Through Print Filters.

5.2.1.3 Paper Types

When defining printers, you can select from predefined paper types or add new paper types from the Form menu. This table describes the paper type definitions available for new paper types:

Paper Type Definitions	Description
Paper Type	Enables you to select the type of paper that the defined printer supports, such as legal, letter, and A4.
Paper Height	Enables you to enter a value that indicates the height of the paper for the selected paper type.
Paper Width	Enables you to enter a value that indicates the width of the paper for the selected paper type.
Unit of Measure	Enables you to enter the unit of measure used to indicate the paper height and width.

The system saves the new paper type and displays it on the Work With Paper Types form. When the definition is complete, the new paper type is available in the grid area of the Printer Setup form. All previous paper type selections are cleared and must be redefined.

5.2.1.4 Default Printers

To set a printer as the default, you must select the Define Default Printer option of the JD Edwards EnterpriseOne Printers application after adding the printer to the system. The printer can be defined as the default printer for:

- A specific version of a report.
- All versions of a specific report.
- All reports.

You can also indicate that the printer is the default printer for a specific user or role, and a specific environment and host.

You must define the default printer as active for the system to recognize it as the default printer. Where multiple printers are configured using the same information, only one printer can be defined as active. If another printer is already defined as the active default, you must change the original default printer to inactive before making

the new printer active. You can perform multiple status changes from the Work With Default Printers form.

5.2.2 Understanding Null Pass-Through Print Filters

The null pass-through print filter enables you to send PDF documents directly to a print queue without converting it to a printer language format. Use null pass-through print filters when you define a custom PDL for a printer.

You can select from available null pass-through print filters or create new filters. You must first select the Custom PDL on the Printer Setup form in the JD Edwards EnterpriseOne Printers application and then select Advanced from the Form menu.

If you are making a copy of a conversion program, the Parameter String field is populated based on the filter that you selected on the Work With Conversion Programs form. Otherwise, the parameter string is entered automatically based on the host from which you are printing (for example ISERIES or HP9000) and the type of printer (postscript, PCL, or line).

This code is an example of the parameter string for null pass-through print filters:

```
-s string_name
```

```
-llibrary_name-f convertPDFToPS
```

Where -s defines the string name, -l defines the library name (this value is the letter l, not the number 1), and -f defines the function name

5.2.3 Forms Used to Add Printers

Form Name	FormID	Navigation	Usage
Printers	W98616S	EnterpriseOne Life Cycle Tools, Report Management, Batch Processing Setup (GH9013), Printers	Add printers, modify printers, and define default printers
Printer Setup Director	W98616AC	Click Add Printer on the Printers form.	Review the printer tasks and begin the Printer Setup Director.
Platform Information	W98616V	Click Next on the Printer Setup Director form.	Enter platform type, printer server name, and print shared name.
Work With Printers	W98616Y	Click Modify Printer on the Printers form.	Select a printer to modify or copy.
Printer Setup	W98616AE	Click Next on the Platform Information form when adding printers. Select a printer and click Select on the Work With Printers form when modifying printers.	Enter or modify the location and model of the printer, paper types, default paper type, and Printer Definition Language.
Work With Default Printers	W98616O	Click Define Default Printer on the Printers form.	Select a printer record.

Form Name	FormID	Navigation	Usage
Default Printer Revisions	W98616M	Click Select on the Work With Default Printers form.	Change the status of a printer.
Work With Conversion Programs	W98616I	Select Custom on the Details tab of the Printer Setup form and then select Advanced from the Form menu.	Select or add a conversion program.
Advanced Conversion Program	W98616J	Click Add on the Work With Conversion Programs form.	Enter a new conversion program name and parameter string.

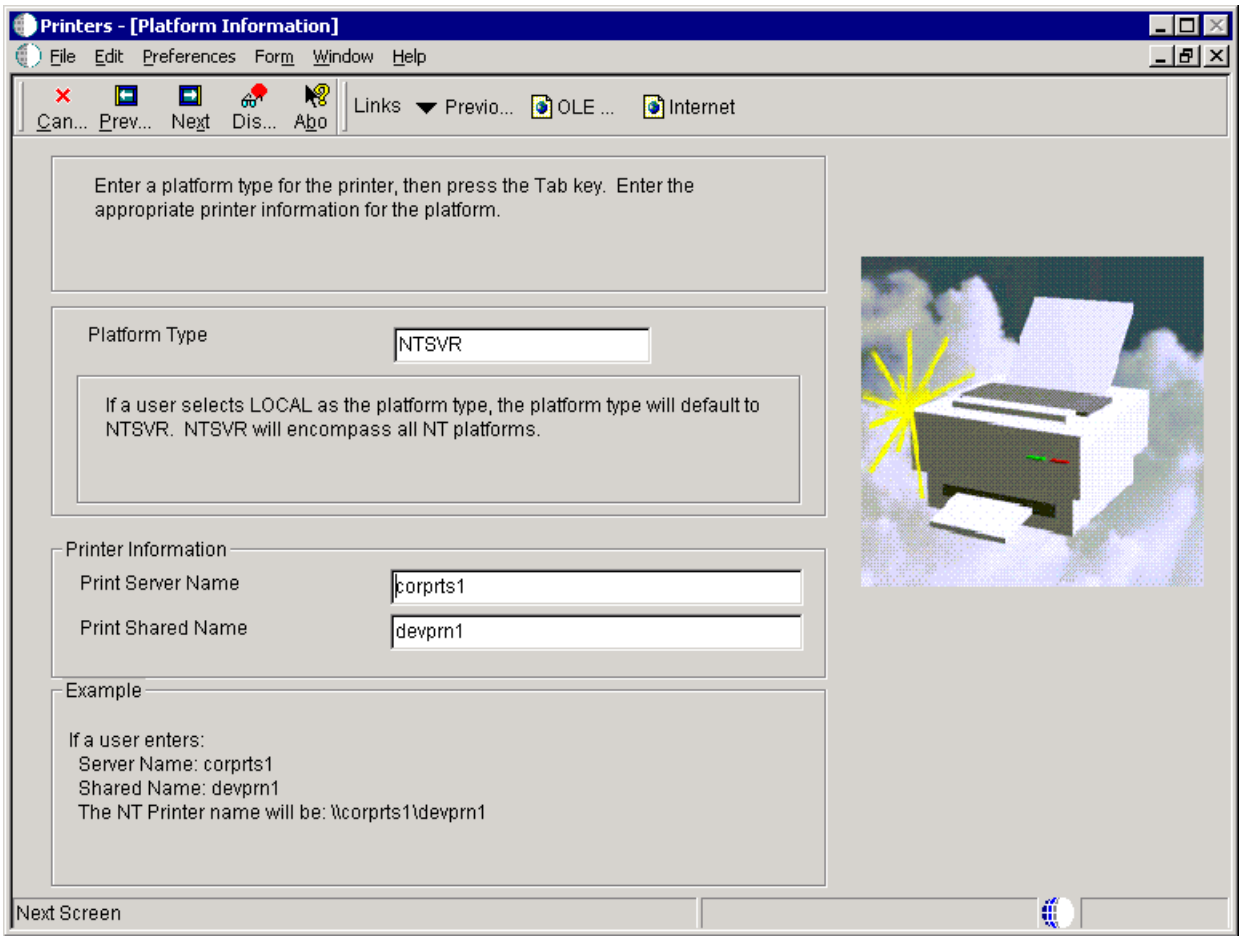
5.2.4 Adding Printers

Access the Printers form.

5.2.4.1 Platform Information

Access the Platform Information form.

Figure 5–1 Platform Information Form



Platform Type
Enter the type of hardware on which the database resides.

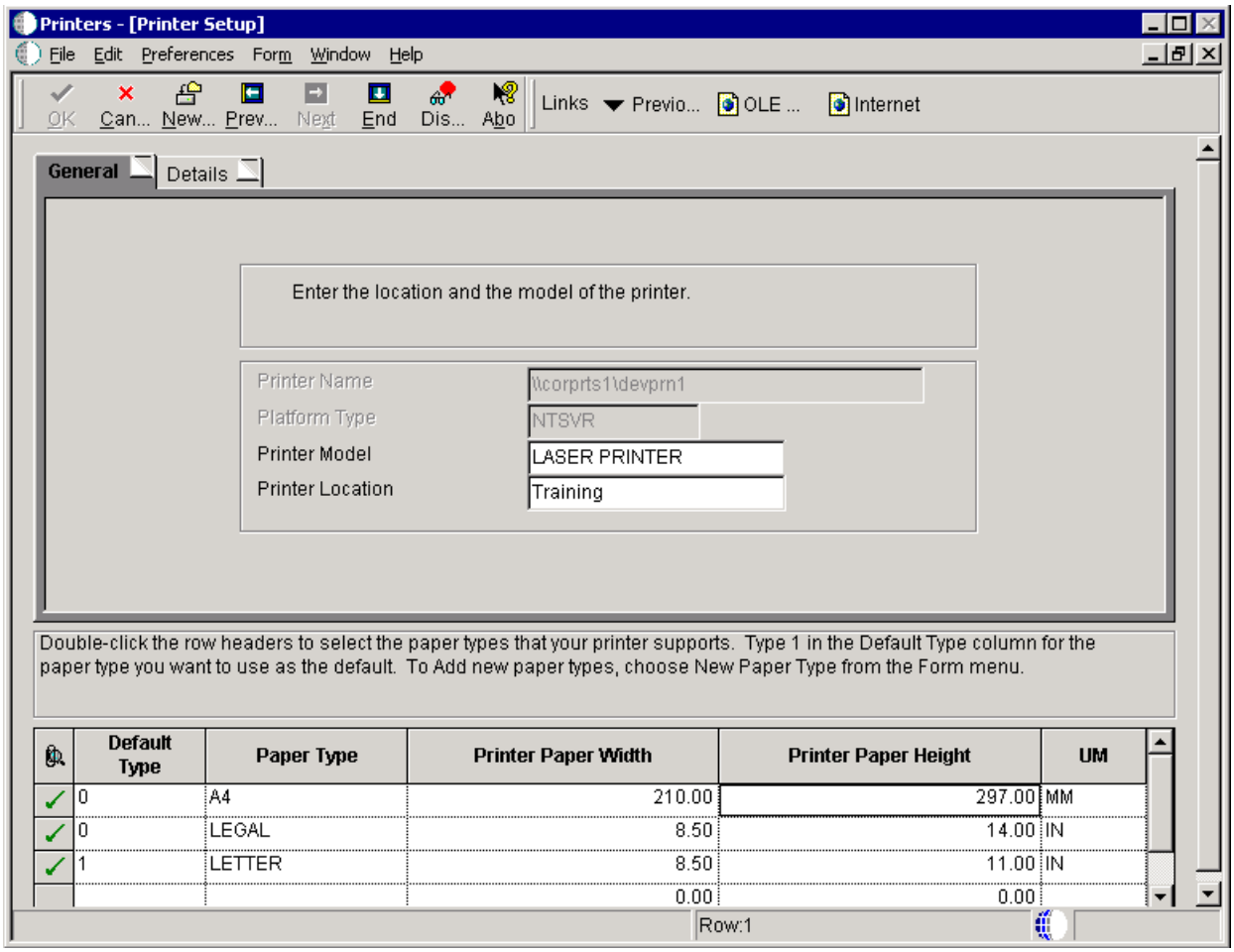
Print Server Name
Enter the name of the machine that receives documents from clients

Print Shared Name
Enter the name of the printer to which the report will be sent.

5.2.4.2 Printer Setup

Access the General tab on the Printer Setup form.

Figure 5–2 Printer Setup Form - General Tab



Printer Name
The system populates this field based on the Print Server Name and the Print Shared Name that you entered on the Platform Information form. This information cannot be modified from this form.

Platform Type
The system populates this field based on the Platform Type that you entered on the Platform Information form. This information cannot be modified from this form.

Printer Model
Enter the model of the printer.

Printer Location
Enter the physical location of the printer.

Attachments
Double-click this field to select the paper type to be used with the defined printer. A check mark appears next to selected paper types.

Default Type
Select a user defined code (UDC) (98 | FP) that indicates the default paper type. Values are:

1: Indicates that the paper type is the default.

0: Indicates that the paper type is available for use with the defined printer but is not the default

Paper Type
Enter the general type of paper that the defined printer supports, such as A4, legal, and letter size.

Printer Paper Width
Enter a value to indicate the width of the paper for the defined paper type. The value is displayed in the unit of measure defined in the UM field.

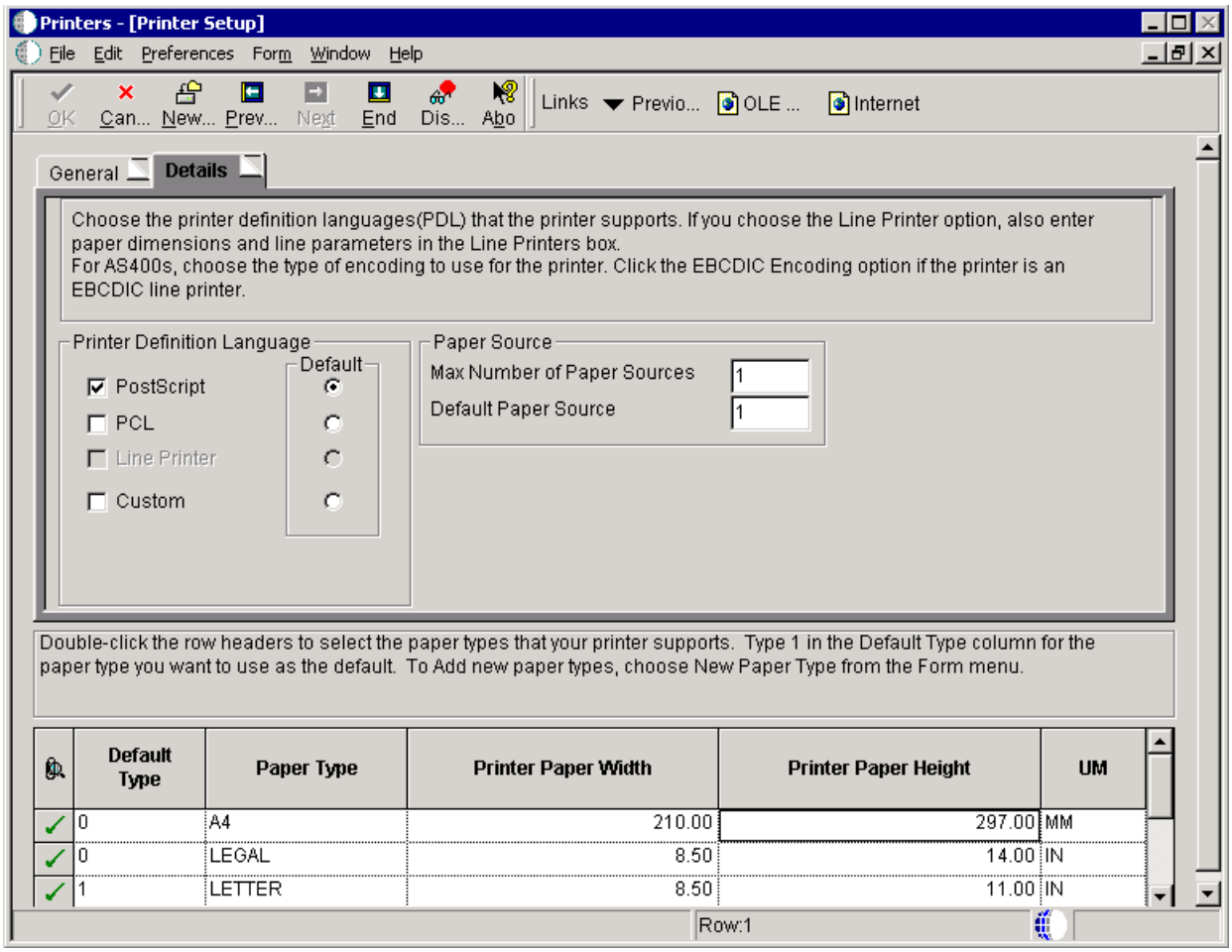
Printer Paper Height
Enter a value to indicate the height of the paper for the defined paper type. The value is displayed in the unit of measure defined in the UM field.

UM
Select a UDC (00 | UM) that indicates the unit of measure that is used to define the width and height of the defined paper type.

5.2.4.3 Printer Setup

Access the Details tab on the Printer Setup form.

Figure 5–3 Printer Setup Form - Details Tab



Printer Definition Language
Select the name of the PDL used by the defined printer. When the PostScript or PCL options are selected, the system disables the Line Printer option.

Multiple PDLs can be selected, but only one can be defined as the default. The PDL can be overridden when batch versions are submitted.

> **Important:** The custom option uses an advanced feature of the JD Edwards EnterpriseOne Printers application. Only users with knowledge of building parameter strings for printers should use this option.

Maximum Number of Paper Sources
Enter the maximum number of paper trays available on the defined printer. This field is only available when you select the PostScript or Custom PDL.

Default Paper Source
Enter the output tray to be used for a specific batch job. This field is only available when the PostScript or Custom PDL is selected.

Characters per Inch
Enter the number of characters per horizontal inch supported by the defined printer. This field is only available when the Line Printer PDL is selected. For an 8 1/2 x 11 inch page the characters per inch value is 10.

Columns per Page
Enter the number of columns per page supported by the defined printer. This field is only available when the Line Printer PDL is selected. For an 8 1/2 x 11 inch page the columns per page value is 85.

Line per Inch
Enter the number of lines per inch supported by the defined printer. This field is only available when the Line Printer PDL is selected. For an 8 1/2 x 11 inch page the lines per inch value is 6.

Line per Page
Enter the number of lines per page supported by the defined printer. This field is only available when the Line Printer PDL is selected. For an 8 1/2 x 11 inch page the lines per page value is 66.

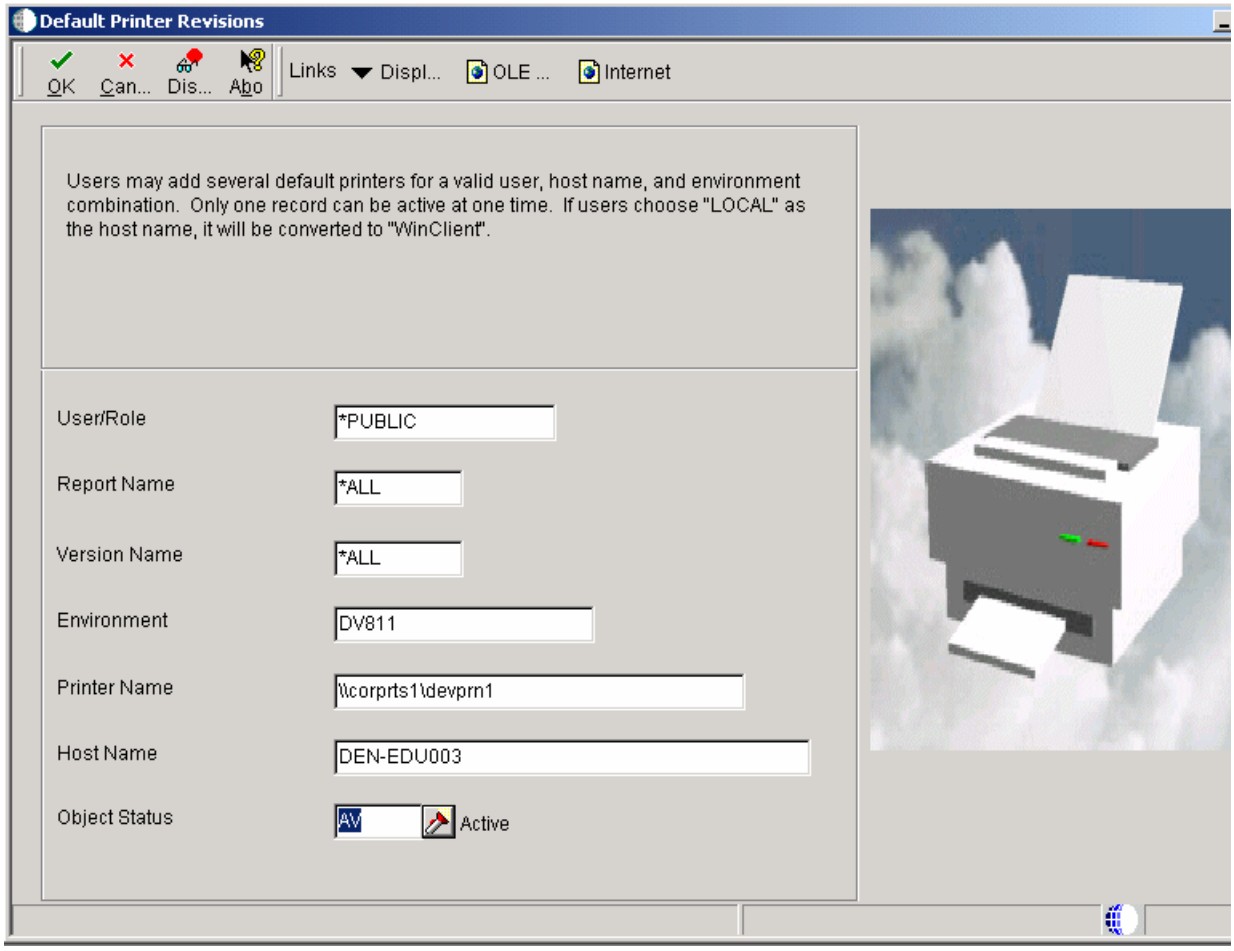
Printer Paper Width
Displays the paper width based on the value that you enter in the Columns per Page field. This field is populated by the system and only appears when the Line Printer PDL is selected.

Printer Paper Height
Displays the paper height based on the value that you enter in the Line per Inch field. This field is populated by the system and only appears when the Line Printer PDL is selected.

5.2.5 Defining Default Printers

Access the Default Printer Revisions form.

Figure 5–4 Default Printer Revisions Form



User/Role
Enter the user ID or role that had permissions to use the printer. *PUBLIC gives permissions to all users.

Report Name
Enter the name of the report that will use this printer. If the field is left blank, the default value is *ALL. *ALL allows all reports to use the printer.

Version Name
Enter the name of the batch version that will use this printer. If the field is left blank, the default value is *ALL. *ALL allows all batch versions to use the printer. If the Report Name is *ALL, the version name defaults to *ALL and is unavailable for input.

Environment
Enter the location of the report and batch version specifications. The system automatically enters the name of the environment that you are currently signed into. Change this information, if necessary.

Printer Name
Enter the name of the printer defined in the JD Edwards EnterpriseOne Printers application to be used as the default.

Host Name
Enter the name of the server that processes the defined batch versions. The visual assist displays the appropriate host names based on the printer name selected.

Object Status
Enter a UDC (H98|ST) that indicates whether the default printer is active or inactive.

5.2.6 Modifying Printers

Access the Work With Printers form.

1. Select the printer that you want to modify and click Select.
2. On the Printer Setup form, modify the information for the printer as necessary.

   You cannot modify the printer name and platform type. If you select a line printer, the paper-type grid at the bottom of the form is disabled.

5.2.7 Copying Printers

Access the Work With Printers form.

1. Select the printer that you want to copy and click Copy.
2. On the Printer Setup form, complete fields as appropriate.
3. Select the Details tab and complete information as appropriate.

5.2.8 Deleting Printers

Access the Work With Printers form.

1. Select a printer, or select multiple printers by holding down the Ctrl key.
2. Click Delete.

   This task removes the printer definition.

5.2.9 Deleting Paper Types

Access the Work With Printers form.

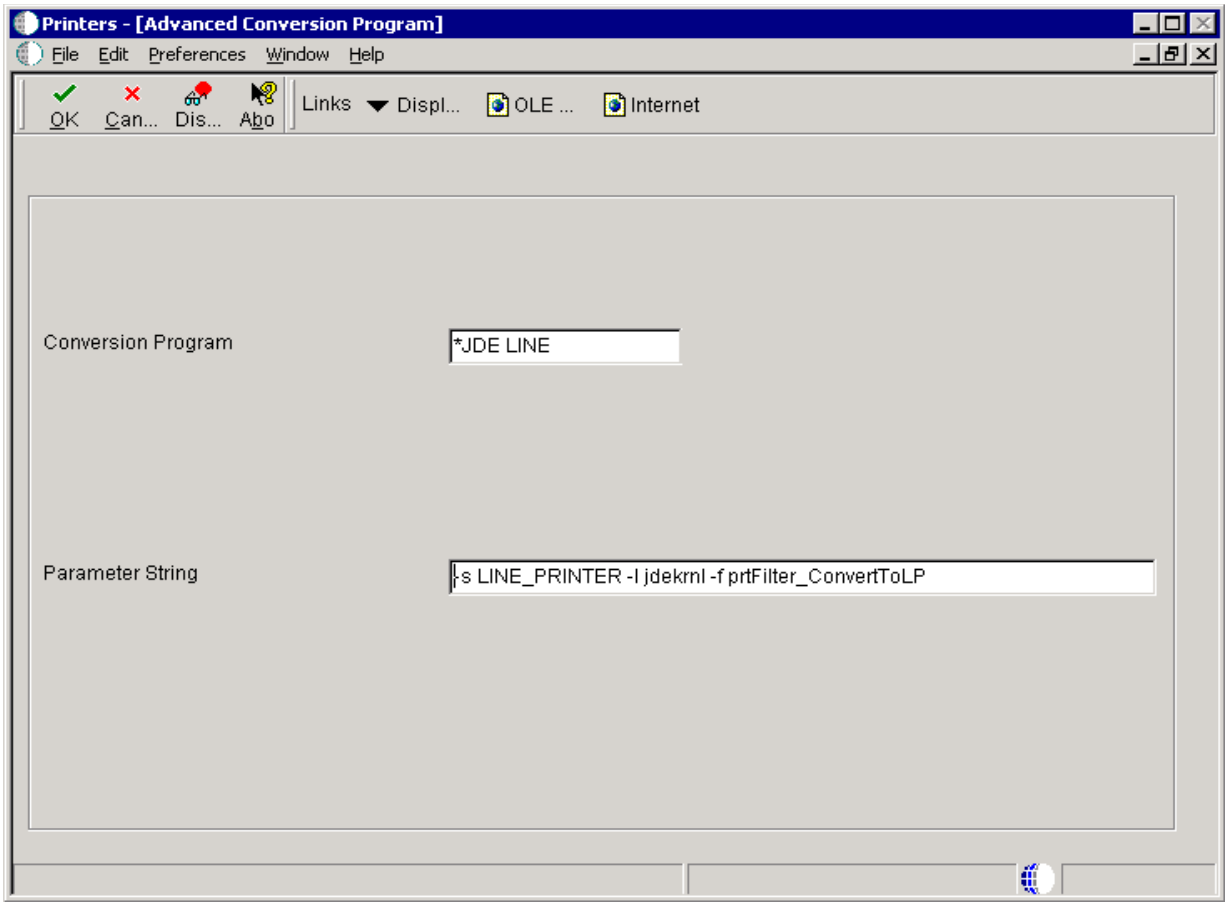
1. Select a printer and click Select.
2. On the Printer Setup form, select New Paper Type from the Form menu.
3. On the Work With Paper Types form, select a paper type and click Delete.
4. On Confirm Delete, click OK.

   The paper type that you deleted no longer displays in the detail area.

5.2.10 Adding Null Pass-through Print Filters

Access the Advanced Conversion Program form.

Figure 5–5 Advanced Conversion Program form



Conversion Program
Select the name of a conversion program (null pass-through print filter) using the visual assist.

Parameter String
Displays the parameter value passed to the conversion program. This value is populated by the system based on the conversion program selected. The parameter string is dependent on the type of printer, and the hardware and software platform being used.

5.2.11 Searching for Incorrect Printer Records

Use this batch process to search the Printer Capability (F986163) table. This report can help you identify incomplete printing records and records that contain incorrect printer information.

From EnterpriseOne Life Cycle Tools, select Report Management (GH9111), Batch Versions.

1. On the Work With Batch Versions - Available Versions form, enter **R9861602** in the Batch Application field and click Find.
2. Run the XJDE0001 version.

   The report lists any printer definition records that might not be complete or that are erroneous and need correction.

3. Using the report, locate the incomplete printer record and correct it.

   See Modifying Printers.

5.3 Setting Up Barcode Fonts

This section provides an overview of barcode fonts and discusses how to:

- Set up printers to use barcode fonts.
- Modify barcode printer information.
- Copy barcode printer information for new printers.
- Delete barcode support information from printers.

5.3.1 Understanding Barcode Fonts

JD Edwards EnterpriseOne supports the use of the BC C39 3 to 1 Medium barcode font and includes this font with the software. After you set up the printers, you can assign a printer to use a barcode font for use with reports. This section describes how to define a printer to support the barcode font BC C39.

Note: Printers that support barcode fonts must use either the PostScript or PCL printer definition languages.

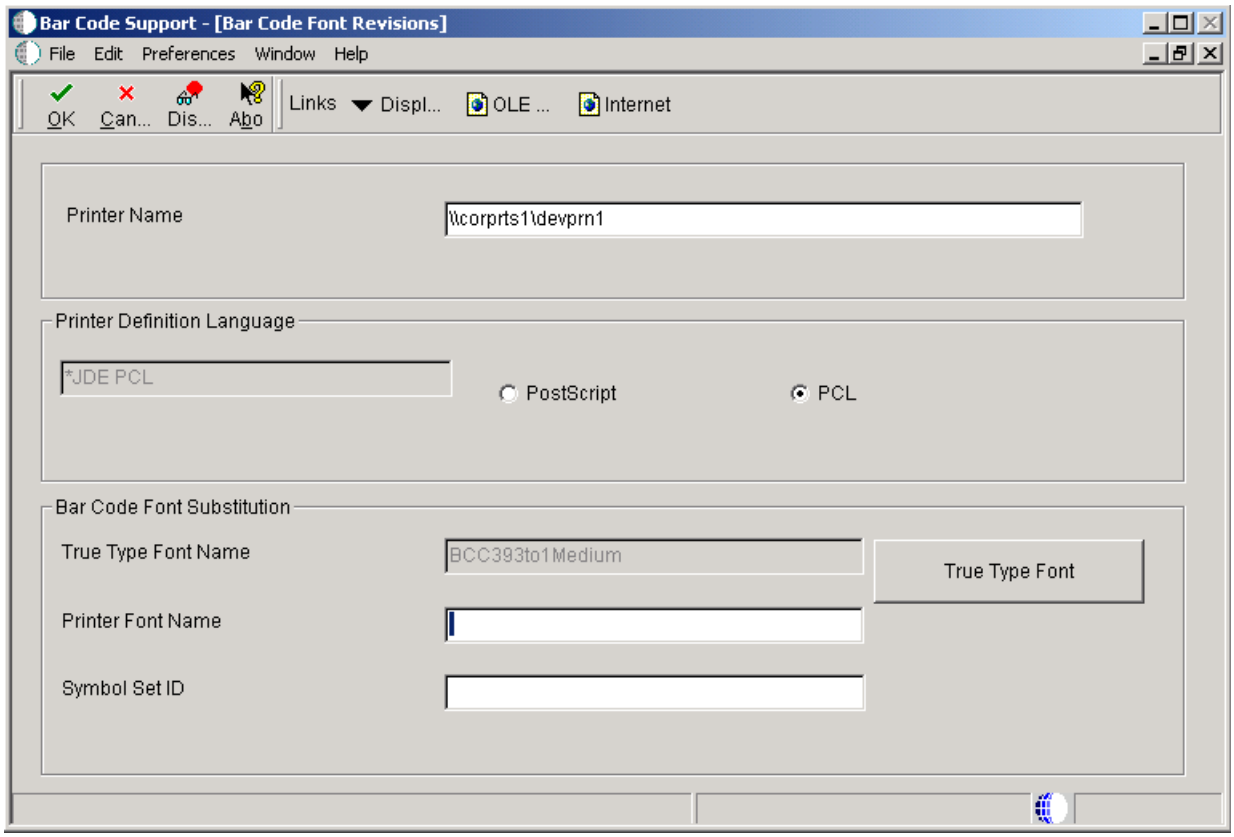
5.3.2 Forms Used to Set Up Printers to Use Barcode Fonts

Form Name	FormID	Navigation	Usage
Work With Bar Code Font	W986166A	EnterpriseOne Life Cycle Tools, Report Management, Batch Processing Setup (GH9013), Bar Code Support	Modify or add printers defined to support barcodes.
Bar Code Support Revisions	W986166B	Click Add on the Work With Bar Code Font form.	Enter the printer name, printer definition language, True Type font, font name, and symbol set ID to be used for barcodes.

5.3.3 Setting Up Printers to Use Barcode Fonts

Access the Bar Code Support Revisions form.

Figure 5–6 Bar Code Font Revisions form



Printer Name
Use the visual assist to select a printer to be used for printing barcode fonts.

Printer Definition Language
Select the printer definition language used by the printer selected. Options include PostScript and PCL.

True Type Font Name
Click the True Type Font button to select the **BC C39 3 to 1 Medium** barcode font. The bar code true type font must reside in the Microsoft Windows font directory.

Printer Font Name
Enter the name of the printer font to be used.

Symbol Set ID
Enter a value that defines the character and character mapping for a specific symbol set. Contact the PCL printer font vendor to obtain this information. This option is available for the PCL printer definition language only.

5.3.4 Modifying Barcode Printer Information

Access the Work With Bar Code Font form.

1. Select the printer for which you want to modify information and click Select.
2. On the Bar Code Font Revisions form, modify the information as appropriate.

5.3.5 Copying Barcode Printer Information for New Printers

Access the Work With Bar Code Font form.

1. Select the printer that you want to copy and click Copy.
2. On the Bar Code Font Revisions form, enter the name of the new printer.
3. Modify information as appropriate.

5.3.6 Deleting Barcode Support Information from Printers

Access the Work With Bar Code Font form.

1. Select the printer for which you want to delete barcode information and click Delete.
2. On the Confirm Delete form, click OK.

5.4 Understanding Multiple Code Sets for PCL

This section discusses:

- Multiple code sets for PCL printing.
- The order of precedence for PCL printing.

5.4.1 Multiple Code Sets for PCL Printing

The JD Edwards EnterpriseOne batch engine generates reports using Unicode data. This means that a single report can contain many different kinds of characters representing multiple languages. However, since PCL printers do not directly support Unicode data, the PCL print filter must translate the report data and tell the printer how to display it.

There are two values that control the translation and display of individual characters in PCL:

- Code page
- Symbol set

You can configure the system to use either a single combination of code page and symbol set, or multiple combinations per report to support an international environment. One of the settings that controls this functionality is the `PRTPCLSymbolSet` setting under the UBE heading of the jde.ini and the jas.ini.

This table describes scenarios that you should consider for defining the `PRTPCLSymbolSet:`

Scenario	Description
The PRTPCLSymbolSet definition is missing or is set to `PRTPCLSymbolSet = *AUTO`.	The PCL print filter automatically analyzes the report data to determine which code page and symbol set values to use to correctly display the characters in the report. The print filter then sends the appropriate values to the printer as it is printing.
The PRTPCLSymbolSet definition is set to `PRTPCLSymbolSet = *NONE`.	A single symbol set is derived based on the code page used to generate the report. This combination of symbol set and code page is used for the entire document.

Scenario	Description
The PRTPCLSymbolSet definition is set to `PRTPCLSymbolSet =XXX`, where XXX represents the desired symbol set.	The defined symbol set is used for all PCL printing.

5.4.2 The Order of Precedence for PCL Printing

The order of precedence determines which code sets or fonts are used when there are multiple print settings defined. PCL printing has two orders of precedence; the first determines code page and the second determines symbol set. The print filter proceeds down the list until a valid value is derived, and then it uses that value.

5.4.2.1 Code Page Order of Precedence

When a print job is sent to a PCL printer, the PCL print filter:

- Checks the Bar Code font support (F986166) table to determine if the report contains barcode information.

  If the report includes barcode information, it prints the report using a barcode character set.

- Checks to determine if multiple code pages are enabled (PRTPCLSymbolSet=*AUTO).

  If multiple code pages are enabled, it analyzes text strings from the report to determine the code page.

- Checks the PRTLocalCodeSet setting and uses the code page indicated by the setting.

- Uses the code page associated with the user's language preference.

5.4.2.2 Symbol Set Order of Precedence

After the code set is determined, the PCL print filter uses rules to determine the symbol set. This table indicates the rules for the PCL print filter:

Print Filter	Rule
Checks to determine if a symbol set is specified in the Bar Code font support table.	If a symbol set is specified, it uses that symbol set.
Checks the PRTPCLSymbolSet setting.	If a specific symbol set is indicated, it uses that set.
Checks the symbol set code associated with the code page.	The print filter selects a symbol set that matches the current code page.

5.5 Designing Reports to Print on Line Printers

This section provides overviews of reports designed to print on line printers and remote iSeries line printers used to print multiple copes of reports, lists the prerequisite, and discusses how to:

- Modify reports to print on line printers.
- Print multiple copies of reports to remote iSeries line printers.

5.5.1 Understanding Reports Designed to Print on Line Printers

When designing reports to print on line printers, you must follow certain guidelines to ensure that the information contained in the reports prints successfully. These guidelines include font family, font size, grid spacing, the width of the fields on the report, paper dimensions, and line parameters:

- Modify the vertical grid alignment.
- Select a fixed-pitch font.

  The Courier New font is a fixed-pitch font and provides the best results; however, you can use other fixed-pitch fonts. For example, for reports that contain text in Japanese, you should use the fixed-pitch version of the MS-Gothic font.
- Modify the font size.
- Modify the field width.

  Since font properties affect the size of the report fields, you might need to adjust field width. You need to override version specifications for a section, specifically the section layout, before you can modify field widths.
- Apply the font selections to the entire report.
- Align fields.

See "Enhancing Reports Using Basic Functionality", "Modifying the Appearance of Report Objects" in the *JD Edwards EnterpriseOne Tools Development Tools: Report Design Aid Guide*.

5.5.2 Understanding Remote iSeries Line Printers Used to Print Multiple Copies of Reports

If the output queue for an iSeries line printer does not support printing multiple copies, the Display Options parameter can be modified to send multiple copies of documents to the remote printer. A system administrator must perform this task, and only for remote output queues.

5.5.3 Prerequisite

Before modifying reports to print on line printers, ensure that you perform one of these steps:

- Create a batch version of a report that will be sent only to line printers.

  Do not make the modifications in the report template because then the report data might not display properly on other printer platforms.
- Check out an existing batch version for modifying to print on line printers.

5.5.4 Modifying Reports to Print on Line Printers

Select a batch version that is checked out and launch JD Edwards EnterpriseOne Report Design Aid.

1. From the Layout menu, select Grid Alignment.
2. On the Alignment Grid form, modify the value in the Vertical field to **16** and click OK.
3. From the File menu, select Report Properties.

4. On the Properties form, select the Font/Color tab, and change the font to **Courier New.**
5. Change the font size to **10.**
6. Select the Apply settings to all objects option and click OK.
7. Widen fields as necessary to provide enough room for the data to display on the report.
8. Align fields as needed using the Align option on the Layout menu.
9. Save the version.

5.5.5 Printing Multiple Copies of Reports to Remote iSeries Line Printers

To print multiple copies to remote iSeries line printers:

1. End the remote writer to which the output queue is connected.
2. Use the Change Output Queue (CHGOUTQ) command to modify the Display Options (DSPOPT) parameter to contain the value **XAIX.**
3. Restart the remote writer.

5.6 Printing Reports

This section discusses:

- Batch versions at submission.
- Batch versions processed on the server.
- Batch versions processed locally on the Microsoft Windows client.
- Print-time characteristics.
- Print settings for batch versions.

5.6.1 Batch Versions at Submission

When you submit batch versions, the batch engine uses a device context to generate a PDF file. This device context includes information such as page size and the printable area of a page. The system generates this information from the printer tables for all platforms.

Within JD Edwards EnterpriseOne, you have the option of viewing the PDF output using Adobe Acrobat Reader, or sending the report directly to a printer. You can also print the report from Adobe Acrobat Reader. When you send the report to a printer, the system uses a conversion filter to transform the PDF file into one of three PDL formats:

- PCL
- PostScript
- Line-printer text

These language formats depend on the type of printer printing the report.

The batch engine uses a logical path to determine to which printer to send reports. If the first method does not return a valid printer name, the batch engine uses the subsequent method.

When you submit batch versions:

1. The batch process triggers the Do Initialize Printer event defined in RDA.

   If this process retrieves a valid printer name, other processes are ignored.

2. You override the default printer name at the time that the report is submitted.

   If you override the default printer with a valid printer name, further processes are ignored.

3. The report specifications pass a printer name to the batch process.

   If this process retrieves a valid printer name, the next process is ignored.

4. The system uses the Printer Definition (F98616) table to determine a valid default printer based on the current user, the environment that the user is signed into, and the host that processes the report.

5.6.2 Batch Versions Processed on the Server

When you submit batch versions to the server, the engine prompts you for a printer name. Valid printer names must have been previously defined by the system administrator. The server automatically creates a PDF file using the settings associated with the selected printer unless event rules override those printer settings. You can affect how the report prints by modifying settings on the Printer Selection form, such as the printer name, page orientation, PDL, and paper type.

When you view the report from the Microsoft Windows client, the system copies the PDF file from the server to the local directory, E812\PrintQueue on the workstation.

When you view the report from the web client, the system copies the PDF file from the server to a temp directory on the workstation. The temp directory is defined in the jas.ini. PDF files are deleted when you sign off of the web client.

In the jas.ini, modify the JDENET setting:

```
[JDENET]

tempFileDir=<temp directory location>
```

Note: When you are using an iSeries platform, the PDF file is stored in the integrated file system (IFS).

See The IFS on the iSeries.

When you run batch versions, you have the option of activating logging capabilities from the Advanced form. If you process batch versions locally from the Microsoft Windows client, the workstation stores the log file in the E812\PrintQueue directory.

If you process batch versions on the server, either from the Microsoft Windows client or the web client, the enterprise server stores the log file in the server PrintQueue directory.

5.6.3 Batch Versions Processed Locally from the Microsoft Windows Client

When you run batch versions locally from a Microsoft Windows client and view the output on the screen, the engine tries to connect to the printer defined in JD Edwards EnterpriseOne Report Design Aid (RDA). If the engine cannot connect, or if there is no printer defined, the engine uses the default printer from the printer tables. Using the settings that it retrieves, the engine creates a PDF file and displays the report through

Adobe Acrobat Reader. The PDF file is stored in the E812\PrintQueue directory on the workstation.

When you run batch versions locally on the Microsoft Windows client and send the output to a printer, the engine displays the Printer Selection form. This form presents you with options to change the printer, page orientation, PDL, paper type, and so on. The initial printer displayed on the Printer Selection form is the one defined in RDA, or the default printer if one was not defined. The engine connects to the printer displayed on the printer form and retrieves the associated settings. Using these settings, the engine:

- Creates a PDF file.
- Converts the PDF into a PDL file using the conversion filter.
- Sends the PDL file to the defined printer.

See "Submitting Batch Versions", "Submitting Batch Versions from the Microsoft Windows Client" in the *JD Edwards EnterpriseOne Tools Development Tools: Batch Versions Guide*.

5.6.4 Print-Time Characteristics

At print time, you can override the printer defined for a report. This is different than overriding the printer when you submit the batch version. At runtime, you can select any valid enterprise printer. At print time, however, you can *only* override the printer with another printer that supports the same platform, PDL, and paper type as the original printer. This is because the batch engine has already created the PDF version of the report and has imbedded the platform, PDL, and paper type information in the PDF file.

5.6.5 Print Settings for Batch Versions

On the Microsoft Windows client, the workstation jde.ini settings control whether reports print immediately and whether the system saves the output after processing the report:

```
[NETWORK QUEUE SETTINGS]
PrintImmediate=TRUE/FALSE
SaveOutput=TRUE/FALSE
```

This table describes the jde.ini settings:

Setting	Description
PrintImmediate	Specifies whether the system automatically prints the report after processing is complete. Values are: TRUE. The system processes the report on the server, generates a PDF file, converts the PDF to the appropriate PDL for the defined printer, and then prints the report. FALSE. The system processes the report on the server, but does not automatically print the report. Users must access the Submitted Job Search form to manually print the report.

Setting	Description
SaveOutput	Specifies whether the system saves or deletes the output after the user views or prints the job. Values are: TRUE. The system saves the output after it has been viewed or printed. FALSE. The system deletes the output after it has been viewed or printed.

The jas.ini settings control whether reports print immediately and whether the system saves the output after processing the report on the web client.

```
[OWWEB]
PrintImmediate=TRUE/FALSE
KeepUBE=TRUE/FALSE
```

This table describes the jas.ini settings:

Setting	Description
PrintImmediate	Specifies whether the system automatically prints the report after processing is complete. Values are: TRUE. The system processes the report on the server, generates a PDF file, converts the PDF to the appropriate PDL for the defined printer, and then prints the report. FALSE. The system processes the report on the server, but does not automatically print the report. Users must access View Job Status–Work With Servers form to manually print the report.
KeepUBE	Specifies whether the system saves or deletes the output after the user views or prints the job. Values are: TRUE. The system saves the output after it has been viewed or printed. FALSE. The system deletes the output after it has been viewed or printed.

6

Working with Output Stream Access

This chapter contains the following topics:

- Section 6.1, "Understanding OSA"
- Section 6.2, "Creating OSA Libraries"
- Section 6.3, "Creating and Associating OSA Interfaces"

6.1 Understanding OSA

You can use OSA interfaces to output to third-party software programs. OSA interfaces enable the third-party program to process and format JD Edwards EnterpriseOne data concurrently. OSA interfaces must be predefined, typically by a representative of the third-party product that you are using. Several interfaces may exist for one program, depending on the section types included in the report and the desired output.

OSA uses its own set of commands from a third-party library.

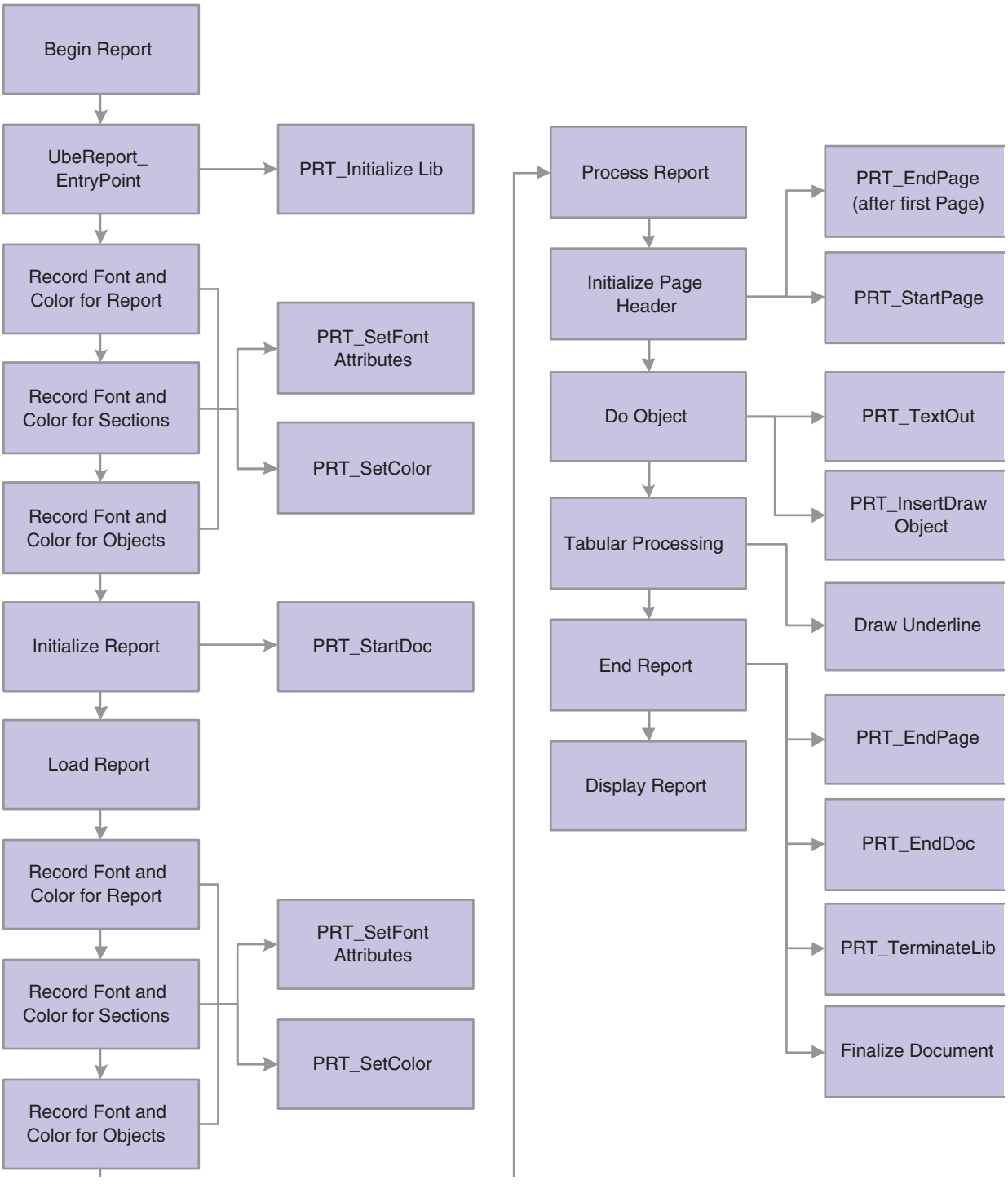
Benefits of using OSA are:

- Employs the formatting options of the target software program.
- Employs the processing power of the target software program.

The system processes the components of batch applications in a specific order. At different points during this processing, you can trigger an event through the OSA interface.

This diagram illustrates the OSA execution points:

Figure 6–1 Execution Points for OSA



At each of these execution points, with the exception of PRT_InitializeLib and PRT_ TerminateLib, you can call an OSA function. You can create functions or you can use existing XML libraries and their functions.

6.2 Creating OSA Libraries

This section discusses:

- OSA libraries
- Function parameters
- OSA documents
- Include files
- File locations and names
- OSASample source code

6.2.1 OSA Libraries

An OSA library is a collection of OSA functions; OSA functions must be included in a library before you can use them. You can create new functions and libraries to satisfy the business requirements.

JD Edwards EnterpriseOne includes an OSA library named OSASample. This OSASample library, along with its source code, serves as an example of how to create an OSA library. The following reference information is provided for developers who need to create their own libraries.

6.2.1.1 Function Signatures

OSA functions are called using the function pointers defined in the JDEOSA file. Therefore, OSA functions should be defined using the same parameters and return values, as in this example set of function prototypes:

```
void MyStartDoc (POSA_REPORT_INFO);
void MySetFont (POSA_REPORT_INFO, POSA_FONT_INFO);
void MySetColor (POSA_REPORT_INFO, unsigned long int);
void MyStartPage (POSA_REPORT_INFO);
void MyTextOut (POSA_REPORT_INFO, POSA_OBJECT_INFO);
void MyDrawObject (POSA_REPORT_INFO, POSA_OBJECT_INFO);
void MyUnderline (POSA_REPORT_INFO, POSA_OBJECT_INFO);
void MyEndPage (POSA_REPORT_INFO, POSA_LINK_INFO, unsigned long);
void MyEndDoc (POSA_REPORT_INFO, POSA_PAGEOF_INFO, unsigned long);
void MyFinalize (POSA_REPORT_INFO);
void MyStartDoc (POSA_REPORT_INFO);
void MySetFont (POSA_REPORT_INFO, POSA_FONT_INFO);
void MySetColor (POSA_REPORT_INFO, unsigned long int);
void MyStartPage (POSA_REPORT_INFO);
void MyTextOut (POSA_REPORT_INFO, POSA_OBJECT_INFO);
void MyDrawObject (POSA_REPORT_INFO, POSA_OBJECT_INFO);
void MyUnderline (POSA_REPORT_INFO, POSA_OBJECT_INFO);
void MyEndPage (POSA_REPORT_INFO, POSA_LINK_INFO, unsigned long);
void MyEndDoc (POSA_REPORT_INFO, POSA_PAGEOF_INFO, unsigned long);
void MyFinalize (POSA_REPORT_INFO);
```

6.2.2 Function Parameters

Function parameters define:

- Start document parameters
- Set font parameters
- Set color parameters
- Start page parameters

- Text out parameters
- Insert draw object parameters
- Draw underline parameters
- End page parameters
- End document parameters
- Finalize document parameters

6.2.2.1 Defining Start Document Parameters

The OSA function that is associated with the Start Document execution point is called by defining these parameters:

```
OSA_REPORT_INFO *pOSAReportInfo
OSA_REPORT_INFO *pOSAReportInfo
```

6.2.2.2 Defining Set Font Parameters

The OSA function that is associated with the Set Font execution point is called by defining these parameters:

```
OSA_REPORT_INFO *pOSAReportInfo,
OSA_FONT_INFO *pOSAFontInfo
```

6.2.2.3 Defining Set Color Parameters

The OSA function that is associated with the Set Color execution point is called by defining these parameters:

```
OSA_REPORT_INFO *pOSAReportInfo,
unsigned long int zColorRef
```

6.2.2.4 Defining Start Page Parameters

The OSA function that is associated with the Start Page execution point is called by defining this parameter:

```
OSA_REPORT_INFO *pOSAReportInfo
```

6.2.2.5 Defining Text Out Parameters

The OSA function that is associated with the Text Out execution point is called by defining these parameters:

```
OSA_REPORT_INFO *pOSAReportInfo,
OSA_OBJECT_INFO *pOSAObjectInfo
```

6.2.2.6 Defining Insert Draw Object Parameters

The OSA function that is associated with the Insert Draw Object execution point is called by defining these parameters:

```
OSA_REPORT_INFO *pOSAReportInfo,
OSA_OBJECT_INFO *pOSAObjectInfo
```

6.2.2.7 Defining Draw Underline Parameters

The OSA function that is associated with the Draw Underline execution point is called by defining these parameters:

```
OSA_REPORT_INFO *pOSAReportInfo,
OSA_OBJECT_INFO *pOSAObjectInfo
```

6.2.2.8 Defining End Page Parameters

The OSA function that is associated with the End Page execution point is called by defining these parameters:

```
OSA_REPORT_INFO pOSAReportInfo,
OSA_LINK_INFO *pOSALinkInfo,
unsigned long ulNumberOfLinks
```

6.2.2.9 Defining End Document Parameters

The OSA function that is associated with the End Document execution point is called by defining these parameters:

```
OSA_REPORT_INFO *pOSAReportInfo,
OSA_PAGEOF_INFO *pOSAPageOfInfo,
unsigned long ulNumberOfPageOf
```

6.2.2.10 Defining Finalize Document Parameters

The OSA function that is associated with the Finalize Document execution point is called by defining this parameter:

```
OSA_REPORT_INFO *pOSAReportInfo
```

6.2.3 OSA Documents

Within the OSAReportInfo structure, the szOSAFileName member enables external applications to specify the name of a file created by OSA functions. A member of the same name is added to the UBEVar structure. After the End Document execution point has been processed, any value that exists in the OSAReportInfo member for szOSAFileName is copied to the corresponding UBEVar member. When a job has finished processing, the UBEVar structure is updated into the Job Control Status Master (F986110) table for that job.

6.2.4 Include Files

The structure and function definitions required for functions interfacing through OSA are contained in the JDEOSA.H file, which is located in the system\include directory. The current contents of this file are:

```
/*****************************************************
Header File Description
* JDEOSA.H Header file to support Output Stream Access functions
*****************************************************
Copyright (c) 1999 - 2004
J.D. Edwards & Company
*This material is proprietary to J.D. Edwards & Company
All rights reserved. The methods and techniques described herein are considered
trade secrets and confidential. Reproduction or distribution, in whole or in
```

```
part, is forbidden except by express written permission of J.D. Edwards & Company.
**********************************************************/
#ifndef JDEOSA_H
#define JDEOSA_H

/** Link Information Structure **/

struct tagOSA_LINK_INFO
{
float fLowerLeftHorizontal;
float fLowerLeftVertical;
float fUpperRightHorizontal;
float fUpperRightVertical;
JCHAR szApplication[11];
JCHAR szForm[11];
JCHAR *szParms;
};

typedef struct tagOSA_LINK_INFO OSA_LINK_INFO, * POSA_LINK_INFO;
/** Font Information **/
struct tagOSA_FONT_INFO
{
long intlfHeight;
long intlfWidth;
long intlfEscapement;
long int lfOrientation;
long int lfWeight;
BYTE lfItalic;
BYTE lfUnderline;
BYTE lfStrikeOut;
BYTElfCharSet;
BYTElfOutPrecision;
BYTElfClipPrecision;
BYTElfQuality;
BYTElfPitchAndFamily;
JCHARlfFaceName[32];
unsigned short nPointSize;
JCHARszAdobeFontName[100];
};
typedef struct tagOSA_FONT_INFO OSA_FONT_INFO, * POSA_FONT_INFO;
/** Item Information **/
struct tagOSA_ITEM_INFO
{
unsigned long ulOccurenceCount;
unsigned long ulRecordFetchCount;
unsigned long ulNumPDFLines;
unsigned shortnReprinting;
unsigned shortnUnderlineThickness;
unsigned shortnUnderlineMargin;
unsigned long int ColorRef;
OSA_FONT_INFO zFontInfo;
unsigned shortnDisplayStyle;
intbPrintMetaData
float fObjectHorizontalPosition;
float fObjectVerticalPosition;
float fObjectEndingHorizontalPosition;
float fObjectEndingVerticalPosition;
float fValueHorizontalPosition;
float fValueVerticalPosition;
float fValueEndingHorizontalPosition;
```

```
float fValueEndingVerticalPosition;
JCHAR *szValue;
JCHAR *szFullText;
};
typedef struct tagOSA_ITEM_INFO OSA_ITEM_INFO, * POSA_ITEM_INFO;
/** Object Information **/
struct tagOSA_OBJECT_INFO
{
JCHAR szDataDictionaryAlias[41];
JCHAR szObjectName[31];
unsigned long idObject;
unsigned longidSection;
unsigned longidRow;
JCHAR szObjectType[3];
unsigned longidLength;
unsigned long idEverestType;
JCHARcDataType;
OSA_ITEM_INFOzOSAItemInfo;
void*pOSASectionInfo;
void*pExternalDataPointer;
JCHARszFutureUse[256];
};
typedef struct tagOSA_OBJECT_INFO OSA_OBJECT_INFO, * POSA_OBJECT_INFO;
/** Section Information Structure **/
struct tagOSA_SECTION_INFO
{
JCHAR*szSectionName;
JCHARszSectionType[50];
shortnSectionType;
JCHARszBusinessViewName[11];
unsigned long idSection;
unsigned long idParentSection;
unsigned long ulNumberOfObjects;
unsigned long ulRecordFetchCount;
OSA_OBJECT_INFO*pOSAObjectInfo;
void*pExternalDataPointer;
JCHARszFutureUse[256];
};
typedef struct tagOSA_SECTION_INFO OSA_SECTION_INFO, * POSA_SECTION_INFO;
/*** Report Information Structure ***/
struct tagOSA_REPORT_INFO
{
JCHARszReport[11];
JCHAR szVersion[11];
JCHAR szMachineKey[16];
JCHAR szEnhv[11];
JCHARszRole[11];
JCHAR szUser[21];
JCHARszHostName[80];
JCHAR szOneWorldRelease[11];
JCHAR szReportTime[12];
JCHAR szDateToday[11];
unsigned int nLocalCodePage;
unsigned int nRemoteCodePage;
int nLocalOperatingSystem;
int nRemoteOperatingSystem;
JCHARszPrinter[256];
unsigned long ulPageSizeVertical;
unsigned long ulPageSizeHorizontal;
unsigned long ulNumberOfCopies;
```

```
unsigned long ulPaperSource;
unsigned short nPageOrientation;
unsigned short nPrinterLinesPerInch;
unsigned short nPrinterCharactersPerInch;
unsigned short nPrinterDefaultFontSize;
JCHAR szPDLProgram[11];
JCHAR szDecimalString[2];
JCHAR cThousandsSeparator;
JCHAR szDateFormat[5];
JCHAR cDateSeparator;
JCHARszLanguage[3];
JCHAR *szReportTitle;
JCHAR szCompanyName[31];
unsigned long ulJobNum;
unsigned long ulCurrentPageNumber;
unsigned long ulActualCurrentPageNumber;
JCHAR szUBEFileName[300];
JCHAR szOSAFileName[256];
JCHARszOSAClientFileName[31];
unsigned long ulNumberOfSections;
OSA_SECTION_INFO *pOSASectionInfo;
void *pExternalDataPointer;
unsigned short *pnLogMessageSeverity;
JCHAR szLogMessage[256];
JCHARszFutureUse[256];
};
typedef struct tagOSA_REPORT_INFO OSA_REPORT_INFO, * POSA_REPORT_INFO;
/** Execution Point Identification Numbers **/
#define OSA_EXPN_START_DOC1
#define OSA_EXPN_SET_FONT 2
#define OSA_EXPN_SET_COLOR3
#define OSA_EXPN_START_PAGE4
#define OSA_EXPN_TEXT_OUT5
#defineOSA_EXPN_DRAW_OBJECT6
#define OSA_EXPN_UNDERLINE7
#define OSA_EXPN_END_PAGE8
#define OSA_EXPN_END_DOC9
#define OSA_EXPN_FINALIZE_DOC 10

/** OSA Function Prototypes **/
#if defined (_WIN32)
#define  OSACDECL _cdecl
#else
#define  OSACDECL
#endif

ypedef void (OSACDECL*FP_OSA_START_DOC) (POSA_REPORT_INFO);
typedef void (OSACDECL*FP_OSA_SET_FONT) (POSA_REPORT_INFO,POSA_FONT_INFO);
typedef void (OSACDECL*FP_OSA_SET_COLOR)(POSA_REPORT_INFO,unsigned long int);
typedef void (OSACDECL*FP_OSA_START_PAGE) (POSA_REPORT_INFO);
typedef void (OSACDECL*FP_OSA_TEXT_OUT) (POSA_REPORT_INFO,POSA_OBJECT_INFO);
typedef void (OSACDECL*FP_OSA_DRAW_OBJECT)(POSA_REPORT_INFO,POSA_OBJECT_INFO);
typedef void (OSACDECL*FP_OSA_UNDERLINE) (POSA_REPORT_INFO,POSA_OBJECT_INFO);
typedef void (OSACDECL*FP_OSA_END_PAGE) (POSA_REPORT_INFO,POSA_LINK_⇒
INFO,unsigned long);
typedef void (OSACDECL*FP_OSA_END_DOC)(POSA_REPORT_INFO);
typedef void (OSACDECL*FP_OSA_FINALIZE_DOC) (POSA_REPORT_INFO);
#endif
```

6.2.5 File Locations and Names

The JD Edwards EnterpriseOne Universal Batch Engine (UBE) performs these steps to load an OSA Library:

1. If the library name, as defined in the OSA Interface Definition (F986169) table, contains a period (.), the UBE ignores the period and any subsequent characters.
2. The UBE adds prefixes, extensions, or both according to the platform on which the UBE is executing.

This table illustrates the prefixes and extensions that are added to the platform on which the UBE is executing:

PLATFORM	EXTENDED LIBRARY NAME
PC	libname + .dll
HPUX	lib + libname + .sl
AIX, SUN	lib + libname + .so
iSeries	libname

The UBE passes the resulting library name to the LoadLibrary function, which uses a standard search strategy to locate the requested library.

6.2.5.1 OSA File Names

The OSA files are generated in the same location as the PDF files, which is the PrintQueue directory. They also follow a similar naming convention as the PDF files, except for the extension. For example, the OSASample output has the extension, .osa.

6.2.6 OSASample Source Code

OSASample includes three components:

- OSAStruct.h
- OSASample.h
- OSASample.c

6.2.6.1 OSAStruct.h

This example shows OSAStruct.h source code:

```
#ifndef OSASAMPLE_DEF_HPP
#define OSASAMPLE_DEF_HPP
#define chAmpersand '&'
#define chOpenAngle '<'
#define chCloseAngle '>'
#define chDoubleQuote ''
#define PAGEOF_TYPE TP
typedef struct tagOSASAMPLE_STRUCT
{
unsigned short nCount;
FILE *fpOutput;
} OSASAMPLE_STRUCT, *POSASAMPLE_STRUCT;
#endif
```

6.2.6.2 OSASample.h

This example shows OSASample.h source code:

```
#ifndef __OSASAMPLE_H__
#define __OSASAMPLE_H__
#include <string.h>
#include <assert.h>
#include <stdio.h>
#include <jdeosa.h>
#if defined (_WIN32)
#undef CDECL
#define CDECL _cdecl
#if defined(IAMOSASAMPLE)
#define APIEXPORT _declspec(dllexport)
#else
#define APIEXPORT _declspec(dllimport)
#endif
#else
#define CDECL
#define APIEXPORT
#endif
#define CLASSEXPORT APIEXPORT
#undef EXTERNC
#if defined(__cplusplus)
#define EXTERNC extern C
#else
#define EXTERNC
#endif
EXTERNC APIEXPORT void CDECL OSASample_StartDoc(POSA_REPORT_INFO
                        pOSAReportInfo);
EXTERNC APIEXPORT void CDECL OSASample_SetFont(POSA_REPORT_INFO
       pOSAReportInfo, POSA_FONT_INFO pOSAFontInfo);
EXTERNC APIEXPORT void CDECL OSASample_SetColor(POSA_REPORT_INFO
       pOSAReportInfo, unsigned long int zColorRef);
EXTERNC APIEXPORT void CDECL OSASample_EndDoc(POSA_REPORT_INFO
                        pOSAReportInfo,
POSA_PAGEOF_INFO pOSAPageofInfo,
unsigned long ulNumberOfStructs);
EXTERNC APIEXPORT void CDECL OSASample_StartPage(POSA_REPORT_INFO
pOSAReportInfo);
EXTERNC APIEXPORT void CDECL OSASample_EndPage(POSA_REPORT_INFO
pOSAReportInfo,
POSA_LINK_INFO pOsaLinkInfo,
unsigned long ulNumberOfLinks);
EXTERNC APIEXPORT void CDECL OSASample_TextOut(POSA_REPORT_INFO
 pOSAReportInfo, POSA_OBJECT_INFO pOSAObjectInfo);
EXTERNC APIEXPORT void CDECL OSASample_DrawObject(POSA_REPORT_INFO
 pOSAReportInfo, POSA_OBJECT_INFO pOSAObjectInfo);
EXTERNC APIEXPORT void CDECL OSASample_DrawUnderLine(POSA_REPORT_INFO
 pOSAReportInfo, POSA_OBJECT_INFO pOSAObjectInfo);
EXTERNC APIEXPORT void CDECL OSASample_FinalizeDoc(POSA_REPORT_INFO
 pOSAReportInfo);
#endif
```

6.2.6.3 OSASAMPLE.c

This example shows OSASample.c source code:

```
#include OSASample.h
#include OSAStruct.h
```

```
/*-------------------------------------------------------------------------
* Function Name: OSA_ReportInfoOut
* Parameters: OSASAMPLE_STRUCT * Pointer to OSA Sample Structure
* OSA_REPORT_INFO * Pointer to Report Info structure
* Exceptions: None
* Return Value: None
* Description: Output data from the Report Info structure
*------------------------------------------------------------------------- */
void OSA_ReportInfoOut(POSASAMPLE_STRUCT pOSAStruct, POSA_REPORT_INFO pOSAReport⇒
Info)
{
/* Check for valid parameter values. If pointers are void, return. */
if (!pOSAStruct || !pOSAStruct->fpOutput || !pOSAReportInfo)
{
return;
}
/* Print values to the output file */
fprintf(pOSAStruct->fpOutput, ****** REPORT INFO ******\n\n);
fprintf(pOSAStruct->fpOutput, Report: %s\n, pOSAReportInfo->szReport);
fprintf(pOSAStruct->fpOutput, Version: %s\n, pOSAReportInfo->szVersion);
fprintf(pOSAStruct->fpOutput, MachineKey: %s\n, pOSAReportInfo->
                      szMachineKey);
fprintf(pOSAStruct->fpOutput, Environment: %s\n, pOSAReportInfo->szEnhv);
fprintf(pOSAStruct->fpOutput, User: %s\n, pOSAReportInfo->szUser);
fprintf(pOSAStruct->fpOutput, Release: %s\n, pOSAReportInfo->
                     szOneWorldRelease);
fprintf(pOSAStruct->fpOutput, Time: %s\n, pOSAReportInfo->szReportTime);
fprintf(pOSAStruct->fpOutput, Date: %s\n, pOSAReportInfo->szDateToday);
fprintf(pOSAStruct->fpOutput, Local Code Page: %d\n, pOSAReportInfo->
                        nLocalCodePage);
fprintf(pOSAStruct->fpOutput, Remote Code Page: %d\n, pOSAReportInfo->
                         nRemoteCodePage);
fprintf(pOSAStruct->fpOutput, Local Operating System: %d\n,
            pOSAReportInfo->nLocalOperatingSystem);
fprintf(pOSAStruct->fpOutput, Remote Operating System: %d\n,
pOSAReportInfo->nRemoteOperatingSystem);
fprintf(pOSAStruct->fpOutput, Printer: %s\n, pOSAReportInfo->szPrinter);
fprintf(pOSAStruct->fpOutput, Page Height: %d\n,
pOSAReportInfo->ulPageSizeVertical);
fprintf(pOSAStruct->fpOutput, Page Width: %d\n,
pOSAReportInfo->ulPageSizeHorizontal);
fprintf(pOSAStruct->fpOutput, Number Of Copies: %d\n,
pOSAReportInfo->ulNumberOfCopies);
fprintf(pOSAStruct->fpOutput, Paper Source: %d\n,
pOSAReportInfo->ulPaperSource);
fprintf(pOSAStruct->fpOutput, Orientation: %d\n,
pOSAReportInfo->nPageOrientation);
fprintf(pOSAStruct->fpOutput, Lines Per Inch: %d\n,
pOSAReportInfo->nPrinterLinesPerInch);
fprintf(pOSAStruct->fpOutput, Default Font Size: %d\n,
pOSAReportInfo->nPrinterDefaultFontSize);
fprintf(pOSAStruct->fpOutput, Printer Type: %s\n,
pOSAReportInfo->szPDLProgram);
fprintf(pOSAStruct->fpOutput, Decimal Separator: %s\n,
pOSAReportInfo->szDecimalString);
fprintf(pOSAStruct->fpOutput, Thousands Separator: %c\n,
pOSAReportInfo->cThousandsSeparator);
fprintf(pOSAStruct->fpOutput, Date Format: %s\n, pOSAReportInfo->
szDateFormat);
fprintf(pOSAStruct->fpOutput, Date Separator: %c\n, pOSAReportInfo->
```

```
cDateSeparator);
fprintf(pOSAStruct->fpOutput, Report Title: %s\n, pOSAReportInfo->
szReportTitle);
fprintf(pOSAStruct->fpOutput, Company Name: %s\n, pOSAReportInfo->
szCompanyName);
fprintf(pOSAStruct->fpOutput, Job Number: %d\n, pOSAReportInfo->ulJobNum);
fprintf(pOSAStruct->fpOutput, Current Page Number: %d\n,
pOSAReportInfo->ulCurrentPageNumber);
fprintf(pOSAStruct->fpOutput, Actual Page Number: %d\n,
pOSAReportInfo->ulActualCurrentPageNumber);
fprintf(pOSAStruct->fpOutput, UBE File Name: %s\n, pOSAReportInfo->
szUBEFileName);
fprintf(pOSAStruct->fpOutput, OSA File Name: %s\n, pOSAReportInfo->
szOSAFileName);
fprintf(pOSAStruct->fpOutput, Number of Sections: %d\n, pOSAReportInfo->
ulNumberOfSections);
fprintf(pOSAStruct->fpOutput, \n****** END REPORT INFO ******\n);
return;
}
/*-------------------------------------------------------------------------
* Function Name: OSA_SectionInfoOut
* Parameters: OSASAMPLE_STRUCT * Pointer to OSA Sample Structure
* OSA_SECTION_INFO * Pointer to Section Info structure
* Exceptions: None
* Return Value: None
* Description: Output data from the Section Info structure
*------------------------------------------------------------------------- */
void OSA_SectionInfoOut(POSASAMPLE_STRUCT pOSAStruct, POSA_SECTION_INFO
pOSASectionInfo)
{
/* Check for valid parameter values. If pointers are void, return. */
if (!pOSAStruct || !pOSAStruct->fpOutput || !pOSASectionInfo)
{
return;
}
fprintf(pOSAStruct->fpOutput, \n\t****** SECTION INFO ******\n\n);
fprintf(pOSAStruct->fpOutput, \tSection Name: %s\n, pOSASectionInfo->
szSectionName);
fprintf(pOSAStruct->fpOutput, \tSection Type: %s\n, pOSASectionInfo->
szSectionType);
fprintf(pOSAStruct->fpOutput, \tBusiness View Name: %s\n, pOSASectionInfo->
szBusinessViewName);
fprintf(pOSAStruct->fpOutput, \tSection ID: %d\n, pOSASectionInfo->
idSection);
fprintf(pOSAStruct->fpOutput, \tParent Section ID: %d\n, pOSASectionInfo->
idParentSection);
fprintf(pOSAStruct->fpOutput, \tNumber of Objects: %d\n, pOSASectionInfo->
ulNumberOfObjects);
fprintf(pOSAStruct->fpOutput, \tRecord Fetch Count: %d\n, pOSASectionInfo->
ulRecordFetchCount);
fprintf(pOSAStruct->fpOutput, \n\t****** END SECTION INFO ******\n);
return;
}
/*-------------------------------------------------------------------------
* Function Name: OSA_ObjectInfoOut
* Parameters: OSASAMPLE_STRUCT * Pointer to OSA Sample Structure
* OSA_OBJECT_INFO * Pointer to Object Info structure
* unsigned short int Flag to control output of Item Info
* Exceptions: None
* Return Value: None
```

```
* Description: Output data from the Object Info and Item Info structures
*------------------------------------------------------------------------- */
void OSA_ObjectInfoOut(POSASAMPLE_STRUCT pOSAStruct,
POSA_OBJECT_INFO pOSAObjectInfo,
unsigned short int nPrintItemInfo)
{
/* Check for valid parameter values. If pointers are void, return. */
if (!pOSAStruct || !pOSAStruct->fpOutput || !pOSAObjectInfo)
{
return;
}
fprintf(pOSAStruct->fpOutput, \n\t\t****** OBJECT INFO ******\n\n);
fprintf(pOSAStruct->fpOutput, \t\tData Dictionary Item: %s\n,
pOSAObjectInfo->szDataDictionaryAlias);
fprintf(pOSAStruct->fpOutput, \t\tObject Name: %s\n, pOSAObjectInfo->
szObjectName);
fprintf(pOSAStruct->fpOutput, \t\tObject ID: %d\n, pOSAObjectInfo->idObject);
fprintf(pOSAStruct->fpOutput, \t\tSection ID: %d\n, pOSAObjectInfo->
idSection);
fprintf(pOSAStruct->fpOutput, \t\tRow ID: %d\n, pOSAObjectInfo->idRow);
fprintf(pOSAStruct->fpOutput, \t\tObject Type: %s\n, pOSAObjectInfo->
szObjectType);
fprintf(pOSAStruct->fpOutput, \t\tObject Length: %d\n, pOSAObjectInfo->
nLength);
fprintf(pOSAStruct->fpOutput, \t\tOneWorld Data Type: %d\n, pOSAObjectInfo->
idEverestType);
fprintf(pOSAStruct->fpOutput, \t\tGeneral Data Type: %c\n, pOSAObjectInfo->
cDataType);
fprintf(pOSAStruct->fpOutput, \n\t\t****** END OBJECT INFO ******\n);
/* Only output Item Info if the parameter indicates to do so */
if (nPrintItemInfo)
{
POSA_ITEM_INFO pOSAItemInfo = &(pOSAObjectInfo->zOSAItemInfo);
fprintf(pOSAStruct->fpOutput, \n\t\t****** ITEM INFO ******\n\n);
fprintf(pOSAStruct->fpOutput, \t\tOccurence Count: %d\n, pOSAItemInfo->
ulOccurenceCount);
fprintf(pOSAStruct->fpOutput, \t\tRecord Fetch Count: %d\n, pOSAItemInfo->
ulRecordFetchCount);
fprintf(pOSAStruct->fpOutput, \t\tNumber Of Lines: %d\n, pOSAItemInfo->
ulNumPDFLines);
fprintf(pOSAStruct->fpOutput, \t\tReprinting: %d\n, pOSAItemInfo->
nReprinting);
fprintf(pOSAStruct->fpOutput, \t\tUnderline Thickness: %d\n, pOSAItemInfo->
nUnderlineThickness);
fprintf(pOSAStruct->fpOutput, \t\tUnderline Margin: %d\n, pOSAItemInfo->
nUnderlineMargin);
fprintf(pOSAStruct->fpOutput, \t\tColor Reference: %d\n, pOSAItemInfo->
ColorRef);
fprintf(pOSAStruct->fpOutput, \t\tFont Face Name: %s\n, pOSAItemInfo->
zFontInfo.lfFaceName);
fprintf(pOSAStruct->fpOutput, \t\tFont Point Size: %d\n, pOSAItemInfo->
zFontInfo.nPointSize);
fprintf(pOSAStruct->fpOutput, \t\tAdobe Font Name: %s\n, pOSAItemInfo->
zFontInfo.szAdobeFontName);
fprintf(pOSAStruct->fpOutput, \t\tDisplay Style: %d\n, pOSAItemInfo->
nDisplayStyle);
fprintf(pOSAStruct->fpOutput, \t\tObject Start X: %f\n, pOSAItemInfo->
fObjectHorizontalPosition);
fprintf(pOSAStruct->fpOutput, \t\tObject Start Y: %f\n, pOSAItemInfo->
fObjectVerticalPosition);
```

```
fprintf(pOSAStruct->fpOutput, \t\tObject End X: %f\n, pOSAItemInfo->
fObjectEndingHorizontalPosition);
fprintf(pOSAStruct->fpOutput, \t\tObject End Y: %f\n, pOSAItemInfo->
fObjectEndingVerticalPosition);
fprintf(pOSAStruct->fpOutput, \t\tValue Start X: %f\n, pOSAItemInfo->
fValueHorizontalPosition);
fprintf(pOSAStruct->fpOutput, \t\tValue Start Y: %f\n, pOSAItemInfo->
fValueVerticalPosition);
fprintf(pOSAStruct->fpOutput, \t\tValue End X: %f\n, pOSAItemInfo->
fValueEndingHorizontalPosition);
fprintf(pOSAStruct->fpOutput, \t\tValue End Y: %f\n, pOSAItemInfo->
fValueEndingVerticalPosition);
fprintf(pOSAStruct->fpOutput, \t\tValue Text: %s\n, pOSAItemInfo->szValue);
fprintf(pOSAStruct->fpOutput, \t\tFull Object Text: %s\n, pOSAItemInfo->
szFullText);
fprintf(pOSAStruct->fpOutput, \n\t\t****** END ITEM INFO ******\n);
}
else
{
fprintf(pOSAStruct->fpOutput, \n\t\t****** No Item Info At This Point ******
\n);
}
return;
}
/*------------------------------------------------------------------------
* Function Name: OSA_LinkInfoOut
* Parameters: OSASAMPLE_STRUCT * Pointer to OSA Sample Structure
* OSA_LINK_INFO * Pointer to array of Link Info structures
* unsigned long The number of elements in the Link Info array
* Exceptions: None
* Return Value: None
* Description: Output data from the Link Info structures, if any.
*------------------------------------------------------------------------ */
void OSA_LinkInfoOut(POSASAMPLE_STRUCT pOSAStruct,
POSA_LINK_INFO pOSALinkInfo,
unsigned long ulNumberOfLinks)
{
unsigned short i =0;
/* Check for valid parameter values. If pointers are void, return. */
if (!pOSAStruct || !pOSAStruct->fpOutput)
{
return;
}
/* Check for valid parameter values. If pointer is void or array is empty,
output a message and return. */
if (!pOSALinkInfo || !ulNumberOfLinks)
{
fprintf(pOSAStruct->fpOutput, \n** No Link Information **\n);
return;
}
fprintf(pOSAStruct->fpOutput, \n****** LINK INFO ******\n\n);
for (i=0; i<ulNumberOfLinks; i++)
{
fprintf(pOSAStruct->fpOutput, Lower Left X: %f\n, pOSALinkInfo[i].
fLowerLeftHorizontal);
fprintf(pOSAStruct->fpOutput, Lower Left Y: %f\n, pOSALinkInfo[i].
fLowerLeftVertical);
fprintf(pOSAStruct->fpOutput, Upper Right X: %f\n, pOSALinkInfo[i].
fUpperRightHorizontal);
fprintf(pOSAStruct->fpOutput, Upper Right Y: %f\n, pOSALinkInfo[i].
```

```
fUpperRightVertical);
fprintf(pOSAStruct->fpOutput, Application Name: %s\n, pOSALinkInfo[i].
szApplication);
fprintf(pOSAStruct->fpOutput, Form Name: %s\n, pOSALinkInfo[i].szForm);
fprintf(pOSAStruct->fpOutput, Parameter String: %s\n\n, pOSALinkInfo[i].
szParms);
}
fprintf(pOSAStruct->fpOutput, \n****** END LINK INFO ******\n);
return;
}
/*-------------------------------------------------------------------------
* Function Name: OSA_FontInfoOut
* Parameters: OSASAMPLE_STRUCT * Pointer to OSA Sample Structure
* OSA_FONT_INFO * Pointer to Font Info structure
* Exceptions: None
* Return Value: None
* Description: Output data from the Font Info structure
*------------------------------------------------------------------------- */
void OSA_FontInfoOut(POSASAMPLE_STRUCT pOSAStruct,
POSA_FONT_INFO pFontInfo)
{
/* Check for valid parameter values. If pointers are void, return. */
if (!pOSAStruct || !pOSAStruct->fpOutput || !pFontInfo)
{
return;
}
fprintf(pOSAStruct->fpOutput, \n****** FONT INFO ******\n\n);
fprintf(pOSAStruct->fpOutput, Font Face Name: %s\n, pFontInfo->lfFaceName);
fprintf(pOSAStruct->fpOutput, Font Point Size: %d\n, pFontInfo->nPointSize);
fprintf(pOSAStruct->fpOutput, Adobe Font Name: %s\n, pFontInfo->
szAdobeFontName);
fprintf(pOSAStruct->fpOutput, \n****** END FONT INFO ******\n);
return;
}
/*-------------------------------------------------------------------------
*⇒
 Function Name: OSA_OpenOutputFile
* Parameters: OSA_REPORT_INFO * Pointer to⇒
 Report Info Structure
* OSASAMPLE_STRUCT * Pointer to OSA Sample Structure
*⇒
 Exceptions: None
* Return Value: None
* Description: Create the file which will⇒
 contain the sample output
*-----------------------------------------------------------⇒
⇒
⇒
⇒
⇒
⇒
⇒
----------------- */
void OSA_OpenOutputFile (POSA_REPORT_INFO pOSAReportInfo,
⇒
POSASAMPLE_STRUCT pOSAStruct)
{
/* Formulate the output file name based on⇒
 information from Report Info. */
strcpy(pOSAReportInfo->szOSAFileName, pOSAReport⇒
```

```
Info->szUBEFileName);
#if defined JDENV_AS400
/* On iSeries, the UBE file name is of⇒
 the form LIBRARY/PRINTQUEUE(F99999),
where 99999 is the job number. We will just⇒
 switch an O for the F to
indicate an OSA file. */
pStrPtr = strrchr( pOSAReportInfo-⇒
⇒
⇒
⇒
⇒
⇒
⇒
>szOSAFileName,'F' );
*pStrPtr = 'O';
#else
/* On platforms other than iSeries, just⇒
 replace the PDF file extension with OSA.⇒
 */
if( !strstr( pOSAReportInfo->sz⇒
OSAFileName,.pdf ) )
{
/* If there is no .pdf extension, just tack on a .osa⇒
 extension */
  strcat( pOSAReportInfo->szOSAFileName, .osa );
}
else
{  sprintf(⇒
 strstr( pOSAReportInfo->szOSAFileName, .pdf ), .osa);
}
#endif
/* Open the OSA file⇒
 for output. */
pOSAStruct->fpOutput= fopen(pOSAReportInfo->szOSAFileName, w+b);
if ⇒
(!pOSAStruct->fpOutput)
{
/* If the file could not be opened, send an error message⇒
 back to the UBE
log */
if (pOSAReportInfo->pnLogMessageSeverity)
{
*(pOSAReportInfo-⇒
>pnLogMessageSeverity) = 1;
}
sprintf(pOSAReportInfo->szLogMessage, Could not⇒
 open OSA file: %s\n,
pOSAReportInfo->szOSAFileName);
return;
}
return;}
/*-------------⇒
⇒
⇒
⇒
⇒
⇒
⇒
------------------------------------------------------------*/
```

```
/* Name: OSASample_⇒
StartDoc */
/* Parameters: OSA_REPORT_INFO* */
/* Exceptions: None */
/* Return⇒
 Value: None */
/* Description: Open the output file, */
/* Output Report, Section⇒
 and Object */
/* properties. */
/*-------------------------------------------------⇒
⇒
⇒
⇒
⇒
⇒
⇒
----------------------*/
EXTERNC APIEXPORT void CDECL OSASample_StartDoc(OSA_⇒
REPORT_INFO*
pOSAReportInfo)
{
POSASAMPLE_STRUCT pOSAStruct = NULL;
POSA_SECTION_INFO⇒
 pOSASectionInfo = NULL;
POSA_OBJECT_INFO pOSAObjectInfo = NULL;
 char *pStrPtr =⇒
 NULL;
unsigned long i = 0;
unsigned long j = 0;
if(!pOSAReportInfo )
{return;
}
/*⇒
 Allocate memory to hold severity value.
Deallocated in OSASample_EndDoc */
if (!p⇒
OSAReportInfo->pnLogMessageSeverity)
{
pOSAReportInfo->pnLogMessageSeverity = malloc⇒
(sizeof( unsigned short));
}
if (pOSAReportInfo->pnLogMessageSeverity)
{
pOSAReport⇒
Info->pnLogMessageSeverity[0] = 0;
}
/* Create the common structure for passing⇒
 values between functions,
if it has not been created before this point. */
if (!p⇒
OSAReportInfo->pExternalDataPointer)
{
pOSAStruct = malloc(sizeof( OSASAMPLE_⇒
STRUCT));
if (pOSAStruct)
{
memset(pOSAStruct, 0, sizeof(OSASAMPLE_STRUCT));
}else
{
```

```
⇒
strcpy(pOSAReportInfo->szLogMessage, OSA: Could not allocate External Data
Pointer.⇒
\n);
/* Set the correct severity to error message severity */
if (pOSAReportInfo->pn⇒
LogMessageSeverity)
{
*(pOSAReportInfo->pnLogMessageSeverity) = (unsigned short)1;
}
}⇒
/* Record the pointer as the external data pointer in Report Info. */
pOSAReport⇒
Info->pExternalDataPointer=pOSAStruct;
}
/* If the external data pointer does not⇒
 exist execution
cannot go on. Set severity to the highest value, assign a message⇒
 for the
log and return */
if(!pOSAReportInfo->pExternalDataPointer)
{
strcpy(p⇒
OSAReportInfo->szLogMessage, OSA: No External Data Pointer at End
Doc.\n);
/* Set⇒
 the correct severity to error message severity */
if (pOSAReportInfo->pnLogMessage⇒
Severity)
{
*(pOSAReportInfo->pnLogMessageSeverity) = (unsigned short)1;
}
return;}
⇒
pOSAStruct=pOSAReportInfo->pExternalDataPointer;
/* Create output file if it has⇒
 not been created yet */
if (!pOSAStruct->fpOutput)
{
OSA_OpenOutputFile (pOSAReport⇒
Info, pOSAStruct);
if (pOSAReportInfo->pnLogMessageSeverity[0] > 0)
return;
}
/*⇒
 Identify the Execution Point */
fprintf(pOSAStruct->fpOutput, ****** START DOC⇒
 EXECUTION POINT ******\n\n);
/* Output Report Info to file. */
OSA_ReportInfoOut(p⇒
OSAStruct, pOSAReportInfo);
/* Output Section Info to file. */
if (!pOSAReportInfo-⇒
>pOSASectionInfo || !pOSAReportInfo->ulNumberOfSections)
{
if (pOSAReportInfo-⇒
>pnLogMessageSeverity)
{
*(pOSAReportInfo->pnLogMessageSeverity) = 2;
}
```

```
sprintf(p⇒
OSAReportInfo->szLogMessage, No Section Info present.\n);
return;
}
pOSASectionInfo =⇒
 pOSAReportInfo->pOSASectionInfo;
for (i=0; i<pOSAReportInfo->ulNumberOfSections;⇒
 i++)
{
OSA_SectionInfoOut(pOSAStruct, pOSASectionInfo);
/* Output Object Info to⇒
 file. */
if (!pOSASectionInfo->pOSAObjectInfo || !pOSASectionInfo->ulNumberOf⇒
Objects)
{
if (pOSAReportInfo->pnLogMessageSeverity)
{
*(pOSAReportInfo->pnLogMessage⇒
Severity) = 3;
}
sprintf(pOSAReportInfo->szLogMessage,
No Object Info present for⇒
 Section %s.\n,
pOSASectionInfo->szSectionName);
return;
}
pOSAObjectInfo = p⇒
OSASectionInfo->pOSAObjectInfo;
for (j=0; j<pOSASectionInfo->ulNumberOfObjects; j++)
⇒
⇒
⇒
⇒
⇒
⇒
⇒
{
OSA_ObjectInfoOut(pOSAStruct, pOSAObjectInfo, 0); /* Do not print Item
Info at⇒
 this time. */
pOSAObjectInfo++;
}
pOSASectionInfo++;
}
}/*----------------------------⇒
⇒
⇒
⇒
⇒
⇒
⇒
⇒
---------------------------------------- */
/* Name: OSASample_EndDoc */
/*⇒
 Parameters: OSA_REPORT_INFO*, OSA_PAGEOF_INFO*, unsigned long */
/* Exceptions:⇒
 None */
/* Return Value: None */
/* Description: Open the output file, */
/* Output⇒
```

```
 Report, Section and Object */
/* properties. */
/*----------------------------------⇒
⇒
⇒
⇒
⇒
⇒
⇒
---------------------------------- */
EXTERNC APIEXPORT void CDECL OSASample_⇒
EndDoc(OSA_REPORT_INFO*
pOSAReportInfo,
OSA_PAGEOF_INFO* pOSAPageofInfo,
unsigned⇒
 long ulNumberOfStructs)
{
POSASAMPLE_STRUCT pOSAStruct = NULL;
/* If OSA does not⇒
 provide the needed parameter (Highly unlikely), then
return */
if(!pOSAReportInfo )
{⇒
⇒
⇒
⇒
⇒
⇒
⇒
return;
}
/* If the external data pointer does not exist execution
cannot go on. Set⇒
 severity to the highest value, assign a message for the
log and return */
if(!p⇒
OSAReportInfo->pExternalDataPointer)
{
strcpy(pOSAReportInfo->szLogMessage, OSA: No⇒
 External Data Pointer at End
Doc.\n);
/* Set the correct severity to error message⇒
 severity */
if (pOSAReportInfo->pnLogMessageSeverity)
{
*(pOSAReportInfo->pnLog⇒
MessageSeverity) = (unsigned short)1;
}
return;
}
/* Close Output File */pOSAStruct=p⇒
OSAReportInfo->pExternalDataPointer;
/* Identify the Execution Point */
fprintf(p⇒
OSAStruct->fpOutput, \n****** END DOC EXECUTION POINT ******);
if (pOSAStruct->fp⇒
Output)
{
fclose (pOSAStruct->fpOutput);
}
```

```
/* Delete the structure created to hold the⇒
 external data */
free (pOSAStruct);
if (pOSAReportInfo->pnLogMessageSeverity)
free(p⇒
OSAReportInfo->pnLogMessageSeverity);
pOSAReportInfo->pExternalDataPointer = NULL;
⇒
return;
}
/*-------------------------------------------------------------------------⇒
 */
/* Name: OSASample_StartPage */
/* Parameters: OSA_REPORT_INFO* */
/* Exceptions:⇒
 None */
/* Return Value: None */
/* Description: Output Report Info to output file.⇒
 */
/* */
/*-------------------------------------------------------------------------⇒
 */
EXTERNC APIEXPORT void CDECL OSASample_StartPage(OSA_REPORT_INFO*
pOSAReportInfo)
⇒
⇒
⇒
⇒
⇒
⇒
⇒
{
POSASAMPLE_STRUCT pOSAStruct = NULL;
/* If OSA does not provide the needed⇒
 parameter (Highly unlikely), then
return */
if(!pOSAReportInfo )
{
return;}
/* If the⇒
 external data pointer does not exist execution
cannot go on. Set severity to the⇒
 highest value, assign a message for the
log and return */
if(!pOSAReportInfo->p⇒
ExternalDataPointer)
{
strcpy(pOSAReportInfo->szLogMessage, OSA: No External Data⇒
 Pointer at
Start Page.\n);
/* Set the correct severity to error message severity */
⇒
if (pOSAReportInfo->pnLogMessageSeverity)
{
*(pOSAReportInfo->pnLogMessageSeverity) ==⇒
⇒
⇒
⇒
⇒
⇒
```

```
⇒
 (unsigned short)1;
}
return;
}
/* Output Report Info */
pOSAStruct=pOSAReportInfo->p⇒
ExternalDataPointer;
/* Check for valid file pointer */
if (!pOSAStruct->fpOutput)
{
⇒
strcpy(pOSAReportInfo->szLogMessage, OSA: No Output File pointer at Start
Page.\n);
⇒
⇒
⇒
⇒
⇒
⇒
⇒
/* Set the correct severity to error message severity */
if (pOSAReportInfo->pnLog⇒
MessageSeverity)
{
*(pOSAReportInfo->pnLogMessageSeverity) = (unsigned short)1;
}
⇒
return;
}
/* Identify the Execution Point */
fprintf(pOSAStruct->fpOutput, \n******⇒
 START PAGE EXECUTION POINT ******\n);
OSA_ReportInfoOut(pOSAStruct, pOSAReport⇒
Info);
return;
}
/*-------------------------------------------------------------------⇒
⇒
⇒
⇒
⇒
⇒
⇒
---- */
/* Name: OSASample_EndPage */
/* Parameters: OSA_REPORT_INFO*, OSA_LINK_⇒
INFO*, unsigned long */
/* Exceptions: None */
/* Return Value: None */
/*⇒
 Description: Output Report Info and Link Info */
/* */
/*--------------------------⇒
⇒
⇒
⇒
⇒
⇒
⇒
```

```
------------------------------------------ */
EXTERNC APIEXPORT void CDECL⇒
 OSASample_EndPage(POSA_REPORT_INFO
pOSAReportInfo,
POSA_LINK_INFO pOSALinkInfo,
⇒
unsigned long ulNumberOfLinks)
{
POSASAMPLE_STRUCT pOSAStruct = NULL;
/* If OSA does⇒
 not provide the needed parameter (Highly unlikely), then
return */
if(!pOSAReport⇒
Info )
{
return;
}
/* If the external data pointer does not exist executioncannot go⇒
 on. Set severity to the highest value, assign a message for the
log and return */
⇒
if(!pOSAReportInfo->pExternalDataPointer)
{
strcpy(pOSAReportInfo->szLogMessage, OSA:⇒
⇒
⇒
⇒
⇒
⇒
⇒
 No External Data Pointer at End
Page.\n);
/* Set the correct severity to error⇒
 message severity */
if (pOSAReportInfo->pnLogMessageSeverity)
{
*(pOSAReportInfo->pn⇒
LogMessageSeverity) = (unsigned short)1;
}
return;
}
/* Output Report Info */⇒
pOSAStruct=pOSAReportInfo->pExternalDataPointer;
/* Check for valid file pointer */
⇒
if (!pOSAStruct->fpOutput)
{
strcpy(pOSAReportInfo->szLogMessage, OSA: No Output⇒
 File pointer at End
Page.\n);
/* Set the correct severity to error message severity⇒
 */
if (pOSAReportInfo->pnLogMessageSeverity)
{
*(pOSAReportInfo->pnLogMessage⇒
Severity) = (unsigned short)1;
}
return;
}
/* Identify the Execution Point */fprintf(p⇒
```

```
OSAStruct->fpOutput, \n****** END PAGE EXECUTION POINT ******\n);
OSA_ReportInfoOut⇒
(pOSAStruct, pOSAReportInfo);
/* Output Link Info */
OSA_LinkInfoOut(pOSAStruct, p⇒
OSALinkInfo, ulNumberOfLinks);
return;
}
/*-----------------------------------------⇒
⇒
⇒
⇒
⇒
⇒
⇒
---------------------------- */
/* Name: OSASample_SetFont */
/* Parameters: OSA_⇒
REPORT_INFO*, OSA_FONT_INFO* */
/* Exceptions: None */
/* Return Value: None */
/*⇒
 Description: Output Font Info */
/* */
/*-----------------------------------------⇒
⇒
⇒
⇒
⇒
⇒
⇒
---------------------------- */
EXTERNC APIEXPORT void CDECL OSASample_SetFont⇒
(POSA_REPORT_INFO
pOSAReportInfo,
POSA_FONT_INFO pOSAFontInfo)
{
POSASAMPLE_STRUCT p⇒
OSAStruct = NULL;
/* If OSA does not provide the needed parameter (Highly unlikely),
⇒
⇒
⇒
⇒
⇒
⇒
⇒
then return */
if(!pOSAReportInfo )
{
return;
}
/* Allocate memory to hold severity⇒
 value.
Deallocated in OSASample_EndDoc */
if (!pOSAReportInfo->pnLogMessageSeverity)
⇒
⇒
⇒
⇒
```

```
⇒
⇒
⇒
{
pOSAReportInfo->pnLogMessageSeverity = malloc(sizeof( unsigned short));
}
if (p⇒
OSAReportInfo->pnLogMessageSeverity)
{
pOSAReportInfo->pnLogMessageSeverity[0] = 0;
}
⇒
/* Create the common structure for passing values between functions,
if it has not⇒
 been created before this point. */
if (!pOSAReportInfo->pExternalDataPointer)
{
⇒
pOSAStruct = malloc(sizeof( OSASAMPLE_STRUCT));
if (pOSAStruct)
{
memset(pOSAStruct,⇒
 0, sizeof(OSASAMPLE_STRUCT));
}
else
{
strcpy(pOSAReportInfo->szLogMessage, OSA:⇒
 Could not allocate External
Data Pointer.\n);
/* Set the correct severity to error⇒
 message severity */
if (pOSAReportInfo->pnLogMessageSeverity)
{
d*(pOSAReportInfo-⇒
>pnLogMessageSeverity) = (unsigned short)1;
}
}
/* Record the pointer as the⇒
 external data pointer in Report Info.*/
pOSAReportInfo->pExternalDataPointer=p⇒
OSAStruct;
}
/* Output Report Info */
pOSAStruct=pOSAReportInfo->pExternalDataPointer;⇒
⇒
⇒
⇒
⇒
⇒
⇒
/* Create output file if it has not been created yet */
if (!pOSAStruct->fpOutput)
{
⇒
OSA_OpenOutputFile (pOSAReportInfo, pOSAStruct);
if (pOSAReportInfo->pnLogMessage⇒
Severity[0] > 0)
return;
}
/* Check for valid file pointer */
```

```
if (!pOSAStruct->fp⇒
Output)
{
strcpy(pOSAReportInfo->szLogMessage, OSA: No Output File pointer at Set
⇒
Font.\n);
/* Set the correct severity to error message severity */
if (pOSAReport⇒
Info->pnLogMessageSeverity)
{
*(pOSAReportInfo->pnLogMessageSeverity) = (unsigned⇒
 short)1;
}
return;
}
/* Identify the Execution Point */
fprintf(pOSAStruct->fpOutput, ⇒
\n****** SET FONT EXECUTION POINT ******\n);
OSA_FontInfoOut(pOSAStruct, pOSAFont⇒
Info);
return;
}
/*-------------------------------------------------------------------⇒
⇒
⇒
⇒
⇒
⇒
⇒
---- */
/* Name: OSASample_SetColor */
/* Parameters: OSA_REPORT_INFO*, unsigned⇒
 long int */
/* Exceptions: None */
/* Return Value: None */
/* Description: Output⇒
 Color Reference Number */
/* */
/*-------------------------------------------------⇒
⇒
⇒
⇒
⇒
⇒
⇒
---------------------- */
EXTERNC APIEXPORT void CDECL OSASample_SetColor(POSA_⇒
REPORT_INFO
pOSAReportInfo,
unsigned long int zColorRef)
{
POSASAMPLE_STRUCT p⇒
OSAStruct = NULL;
/* If OSA does not provide the needed parameter (Highly unlikely),
⇒
⇒
⇒
⇒
⇒
⇒
```

```
⇒
then return */
if(!pOSAReportInfo )
{
return;
}
/* If the external data pointer does⇒
 not exist execution
cannot go on. Set severity to the highest value, assign a⇒
 message for the
log and return */
if(!pOSAReportInfo->pExternalDataPointer)
{
strcpy⇒
(pOSAReportInfo->szLogMessage, OSA: No External Data Pointer at Set
Color.\n);
/*⇒
 Set the correct severity to error message severity */
if (pOSAReportInfo->pnLog⇒
MessageSeverity)
{
*(pOSAReportInfo->pnLogMessageSeverity) = (unsigned short)1;
}
⇒
return;
}
/* Output Report Info */
pOSAStruct=pOSAReportInfo->pExternalDataPointer;
/*⇒
 Check for valid file pointer */
if (!pOSAStruct->fpOutput)
{
strcpy(pOSAReportInfo-⇒
>szLogMessage, OSA: No Output File pointer at Set
Color.\n);
/* Set the correct⇒
 severity to error message severity */
if (pOSAReportInfo->pnLogMessageSeverity)
{
*⇒
(pOSAReportInfo->pnLogMessageSeverity) = (unsigned short)1;
}
return;
}
/* Identify⇒
 the Execution Point */
fprintf(pOSAStruct->fpOutput, \n****** SET COLOR: %d ******⇒
\n, zColorRef);
return;
}
/*-------------------------------------------------------------⇒
⇒
⇒
⇒
⇒
⇒
⇒
--------------- */
/* Name: OSASample_TextOut */
/* Parameters: OSA_REPORT_INFO*,⇒
```

```
 OSA_OBJECT_INFO* */
/* Exceptions: None */
/* Return Value: None */
/* Description:⇒
 Output Font Info */
/* */
/*-------------------------------------------------------⇒
⇒
⇒
⇒
⇒
⇒
⇒
----------------- */
EXTERNC APIEXPORT void CDECL OSASample_TextOut(POSA_REPORT_⇒
INFO
pOSAReportInfo,
POSA_OBJECT_INFO pOSAObjectInfo)
{
POSASAMPLE_STRUCT pOSAStruct =⇒
 NULL;
/* If OSA does not provide the needed parameter (Highly unlikely),
then⇒
 return */
if(!pOSAReportInfo )
{
return;
}
/* If the external data pointer does not⇒
 exist execution
cannot go on. Set severity to the highest value, assign a message⇒
 for the
log and return */
if(!pOSAReportInfo->pExternalDataPointer)
{
strcpy(p⇒
OSAReportInfo->szLogMessage, OSA: No External Data Pointer at Text
Out.\n);
/* Set⇒
 the correct severity to error message severity */
if (pOSAReportInfo->pnLogMessage⇒
Severity)
{
*(pOSAReportInfo->pnLogMessageSeverity) = (unsigned short)1;
}
return;}
/*⇒
 Output Report Info */
pOSAStruct=pOSAReportInfo->pExternalDataPointer;
/* Check for⇒
 valid file pointer */
if (!pOSAStruct->fpOutput)
{
strcpy(pOSAReportInfo->szLog⇒
Message, OSA: No Output File pointer at Text
Out.\n);
/* Set the correct severity to⇒
 error message severity */
if (pOSAReportInfo->pnLogMessageSeverity)
{
```

```
*(pOSAReport⇒
Info->pnLogMessageSeverity) = (unsigned short)1;
}
return;
}
/* Identify the Execution⇒
 Point */
fprintf(pOSAStruct->fpOutput, \n****** TEXT OUT EXECUTION POINT ******\n);
⇒
⇒
⇒
⇒
⇒
⇒
⇒
OSA_ObjectInfoOut(pOSAStruct, pOSAObjectInfo, 1);
return;
}
/*------------------------⇒
⇒
⇒
⇒
⇒
⇒
⇒
---------------------------------------------- */
/* Name: OSASample_Underline */
⇒
/* Parameters: OSA_REPORT_INFO*, OSA_OBJECT_INFO* */
/* Exceptions: None */
/*⇒
 Return Value: None */
/* Description: Output Font Info */
/* */
/*--------------------⇒
⇒
⇒
⇒
⇒
⇒
⇒
---------------------------------------------------- */
EXTERNC APIEXPORT void⇒
 CDECL OSASample_DrawUnderline(POSA_REPORT_INFO
pOSAReportInfo,
POSA_OBJECT_INFO p⇒
OSAObjectInfo)
{
POSASAMPLE_STRUCT pOSAStruct = NULL;
/* If OSA does not provide the⇒
 needed parameter (Highly unlikely), then return */
if(!pOSAReportInfo )
{
return;
}/*⇒
 If the external data pointer does not exist execution
cannot go on. Set severity⇒
 to the highest value, assign a message for the
log and return */
if(!pOSAReportInfo-⇒
```

```
⇒
⇒
⇒
⇒
⇒
⇒
>pExternalDataPointer)
{
strcpy(pOSAReportInfo->szLogMessage, OSA: No External Data⇒
 Pointer at
Draw Underline.\n);
/* Set the correct severity to error message⇒
 severity */
if (pOSAReportInfo->pnLogMessageSeverity)
{
*(pOSAReportInfo->pnLog⇒
MessageSeverity) = (unsigned short)1;
}
return;
}
/* Output Report Info */pOSAStruct=p⇒
OSAReportInfo->pExternalDataPointer;
/* Check for valid file pointer */
if (!p⇒
OSAStruct->fpOutput)
{
strcpy(pOSAReportInfo->szLogMessage, OSA: No Output File⇒
 pointer at Draw
Underline.\n);
/* Set the correct severity to error message⇒
 severity */
if (pOSAReportInfo->pnLogMessageSeverity)
{
*(pOSAReportInfo->pnLog⇒
MessageSeverity) = (unsigned short)1;
}
return;
}
/* Identify the Execution Point */⇒
fprintf(pOSAStruct->fpOutput, \n****** DRAW UNDERLINE EXECUTION POINT
******\n);
OSA_⇒
⇒
⇒
⇒
⇒
⇒
⇒
ObjectInfoOut(pOSAStruct, pOSAObjectInfo, 1);
return;
}
/*---------------------------⇒
------------------------------------------ */
/* Name: OSASample_DrawObject */
/*⇒
 Parameters: OSA_REPORT_INFO*, OSA_OBJECT_INFO* */
/* Exceptions: None */
/* Return⇒
 Value: None */
/* Description: Output Font Info */
```

```
/* */
/*-------------------------⇒
⇒
⇒
⇒
⇒
⇒
⇒
------------------------------------------ */
EXTERNC APIEXPORT void CDECL⇒
 OSASample_DrawObject(POSA_REPORT_INFO
pOSAReportInfo,
POSA_OBJECT_INFO pOSAObject⇒
Info)
{
POSASAMPLE_STRUCT pOSAStruct = NULL;
/* If OSA does not provide the needed⇒
 parameter (Highly unlikely),
then return */
if(!pOSAReportInfo )
{
return;}
/* If the⇒
 external data pointer does not exist execution
cannot go on. Set severity to the⇒
 highest value, assign a message for the
log and return */
if(!pOSAReportInfo->p⇒
ExternalDataPointer)
{
strcpy(pOSAReportInfo->szLogMessage, OSA: No External Data⇒
 Pointer at
Draw Object.\n);
/* Set the correct severity to error message severity */
⇒
⇒
⇒
⇒
⇒
⇒
⇒
if (pOSAReportInfo->pnLogMessageSeverity)
{
*(pOSAReportInfo->pnLogMessageSeverity) =⇒
⇒
⇒
⇒
⇒
⇒
⇒
 (unsigned short)1;
}
return;
}
/* Output Report Info */
pOSAStruct=pOSAReportInfo->p⇒
ExternalDataPointer;
/* Check for valid file pointer */
if (!pOSAStruct->fpOutput)
{
```

```
⇒
strcpy(pOSAReportInfo->szLogMessage, OSA: No Output File pointer at
Draw Object.\n);
⇒
⇒
⇒
⇒
⇒
⇒
⇒
/* Set the correct severity to error message severity */
if (pOSAReportInfo->pnLog⇒
MessageSeverity)
{
*(pOSAReportInfo->pnLogMessageSeverity) = (unsigned short)1;
}
⇒
return;
}
/* Identify the Execution Point */
fprintf(pOSAStruct->fpOutput, \n******⇒
 DRAW OBJECT EXECUTION POINT ******\n);
OSA_ObjectInfoOut(pOSAStruct, pOSAObject⇒
Info, 1);
return;
}
```

6.3 Creating and Associating OSA Interfaces

This section provides an overview of OSA interfaces and discusses how to:

- Create OSA interface definitions.
- Associate an OSA interface with an object.

6.3.1 Understanding OSA Interfaces

Before reports can be output using OSA, you must define the interface. Typically, once defined, OSA interfaces are associated with specific reports or batch versions; however, you can override the default OSA at runtime. You can select from any valid OSA at runtime, although the results might vary depending on how robust the OSA. OSA interfaces can be associated in the same way as default printers with:

- Environments
- Hosts
- Users or roles

Depending on the report output requirements, you might need to define multiple interfaces.

6.3.1.1 Using the System Hierarchy to Resolve Priority Conflicts

The general hierarchy that the system uses to resolve OSA interfaces that are associated with more than one report or version is illustrated in the following table. The system uses the same hierarchy for each user or role, processing in this order:

1. Username

2. Role
3. *PUBLIC

Report	Version	Environment	Host
report	version	environment	hosttype
report	version	environment	*ALL
report	version	*ALL	hosttype
report	version	*ALL	*ALL
report	*ALL	environment	hosttype
report	*ALL	environment	*ALL
report	*ALL	*ALL	hosttype
report	*ALL	*ALL	*ALL
*ALL	*ALL	environment	hosttype
*ALL	*ALL	environment	*ALL
*ALL	*ALL	*ALL	hosttype
*ALL	*ALL	*ALL	*ALL

6.3.2 Forms Used to Create and Associate OSA Interfaces

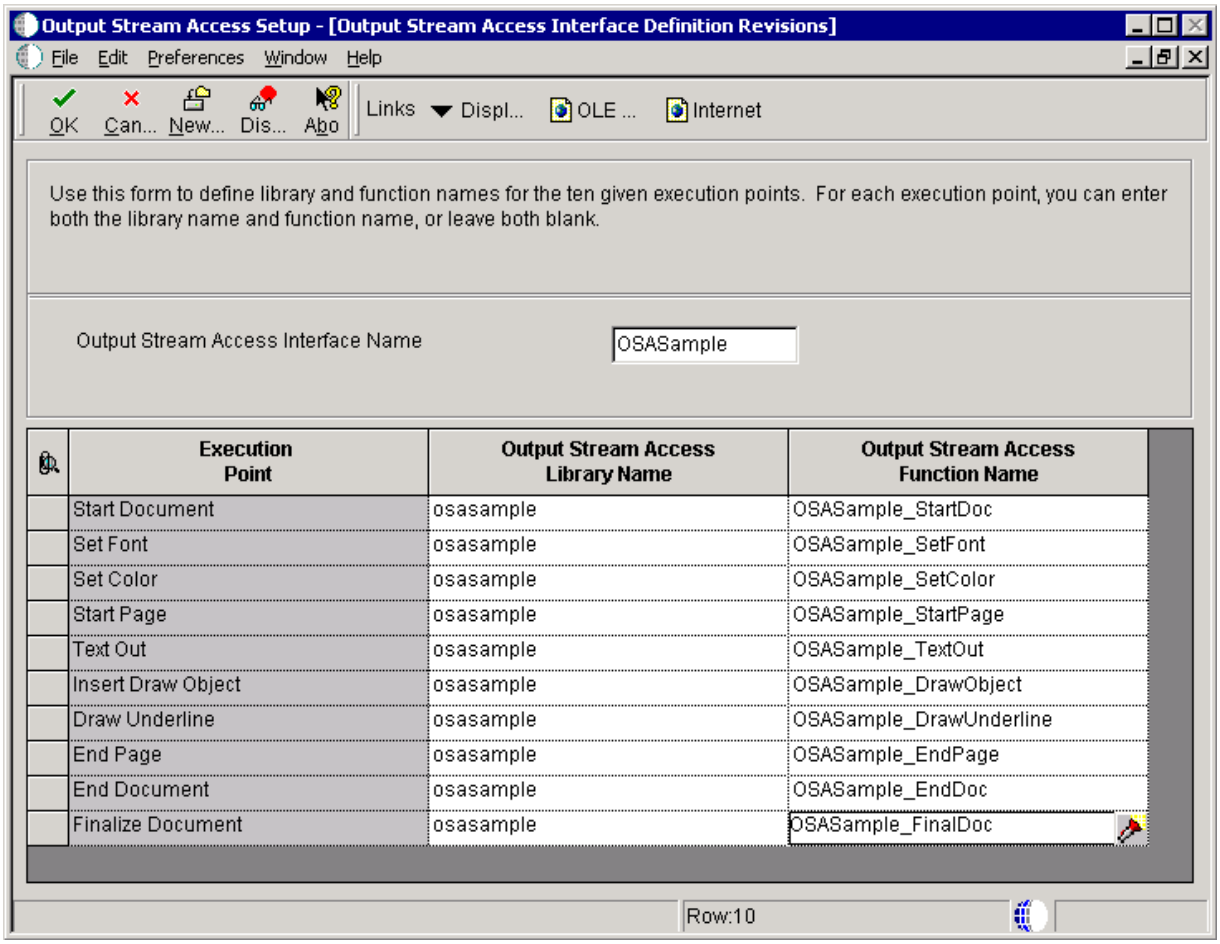
Form Name	FormID	Navigation	Usage
Output Stream Access Setup	W986168F	EnterpriseOne Life Cycle Tools, Report Management, Batch Processing Setup (GH9013), Output Stream Access Setup	Add or modify OSA interface definitions and add or modify OSA usage specifications.
Work With Output Stream Access Interface Definition	W986168C	Click Add or modify the Output Stream Access Interface Definition on the Output Stream Access Setup form.	Add or select an OSA interface definition.
Output Stream Access Interface Definition Revisions	W986168I	Click Add on the Work With Output Stream Access Interface Definition form.	Enter the OSA interface name and, for each execution point, enter the OSA library names and OSA function names.
Work With Output Stream Access Interface Usage	W986168D	Click Add or modify the Output Stream Access Interface Usages specification on the Output Stream Access Setup form.	Add or select an OSA interface usage.

Form Name	FormID	Navigation	Usage
Output Stream Access Interface Usage Revisions	W986168M	Click Add on the Work With Output Stream Access Interface Usage form.	Enter an OSA interface name and the report, version, environment, host, user or role, and usage status associated with the OSA interface.

6.3.3 Creating OSA Interface Definitions

Access the Output Stream Access Interface Definition Revisions form.

Figure 6–2 Output Stream Access Interface Definition Revisions Form



Output Stream Access Interface Name

Enter a unique name that identifies a set of external functions that can receive and process information during execution. JD Edwards EnterpriseOne OSA interfaces begin with the letters JDE. It is recommended that you do not begin custom interface names with JDE.

Output Stream Access Library Name

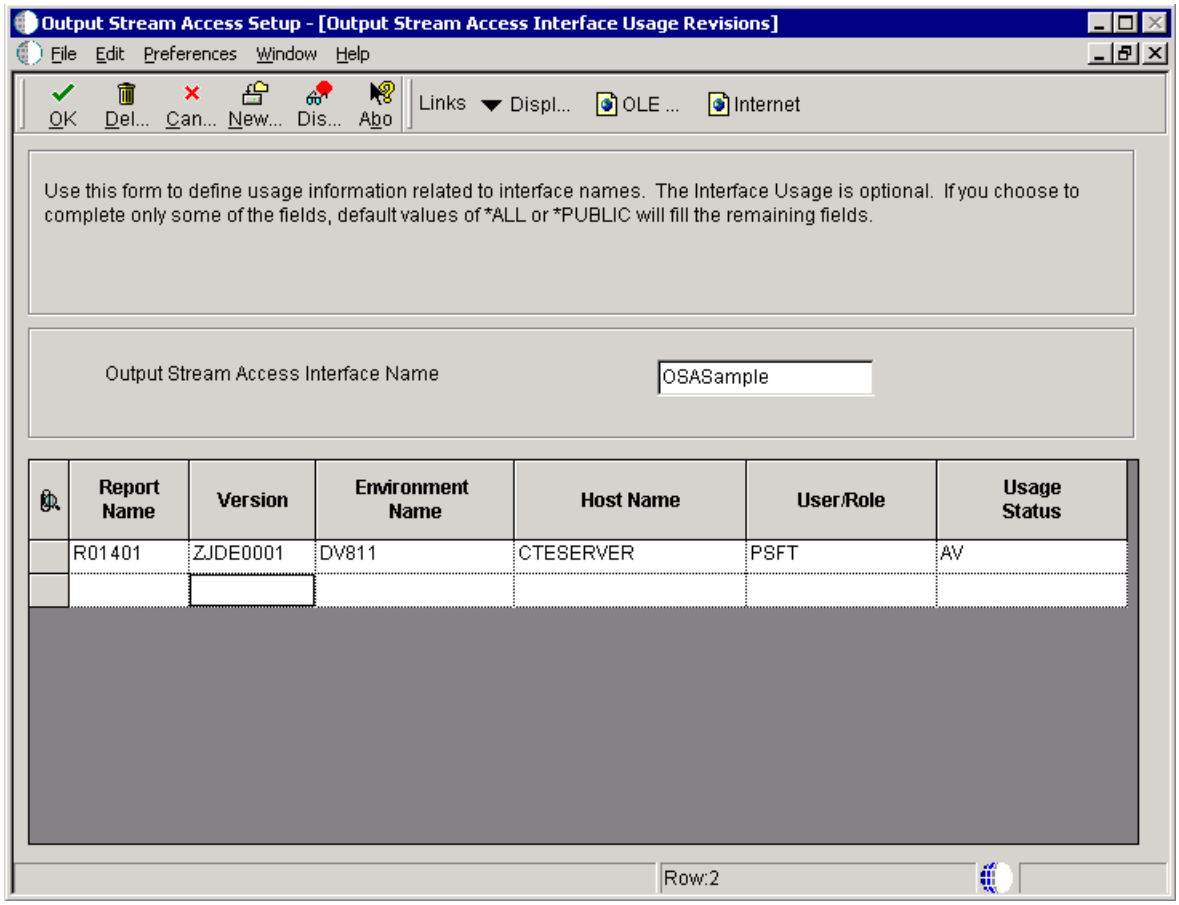
Enter the shared libraries that contain functions that use the data provided through the OSA interface.

Output Stream Access Function Name
Enter the functions that conform with the parameters and calling conventions of the OSA interface. The system executes the OSA functions at execution points following a set of parameters. Execution points with no associated function are ignored when the OSA interface executes.

6.3.4 Associating an OSA Interface with an Object

Access the Output Stream Access Interface Usage Revisions form.

Figure 6–3 Output Stream Access Interface Usage Revisions Form



Output Stream Access Interface Name
Enter the OSA interface name to associate with an object. Use the visual assist to select a valid interface name.

Report Name
Enter the name of the report to associate with the OSA interface. *ALL indicates all reports.

Version
Enter the name of the batch version to associate with the OSA interface. *ALL indicates all batch versions of the defined report.

Environment Name
Enter the location of the report and batch version specifications.

Host Name
Enter the name of the server that processes the defined batch version.

User/Role
Enter the user ID or role with permissions to use the OSA interface. *PUBLIC gives permissions to all users.

Usage Status
Select a user-defined code (UDC) (H98|ST) that indicates whether the OSA interface is active or not active.

Glossary

Accessor Methods/Assessors

Java methods to “get” and “set” the elements of a value object or other source file.

activity rule

The criteria by which an object progresses from one given point to the next in a flow.

add mode

A condition of a form that enables users to input data.

Advanced Planning Agent (APAg)

A JD Edwards EnterpriseOne tool that can be used to extract, transform, and load enterprise data. APAg supports access to data sources in the form of rational databases, flat file format, and other data or message encoding, such as XML.

application server

Software that provides the business logic for an application program in a distributed environment. The servers can be Oracle Application Server (OAS) or WebSphere Application Server (WAS).

Auto Commit Transaction

A database connection through which all database operations are immediately written to the database.

batch processing

A process of transferring records from a third-party system to JD Edwards EnterpriseOne.

In JD Edwards EnterpriseOne Financial Management, batch processing enables you to transfer invoices and vouchers that are entered in a system other than JD Edwards EnterpriseOne to JD Edwards EnterpriseOne Accounts Receivable and JD Edwards EnterpriseOne Accounts Payable, respectively. In addition, you can transfer address book information, including customer and supplier records, to JD Edwards EnterpriseOne.

batch server

A server that is designated for running batch processing requests. A batch server typically does not contain a database nor does it run interactive applications.

batch-of-one

A transaction method that enables a client application to perform work on a client workstation, then submit the work all at once to a server application for further processing. As a batch process is running on the server, the client application can continue performing other tasks.

best practices

Non-mandatory guidelines that help the developer make better design decisions.

BPEL

Abbreviation for Business Process Execution Language, a standard web services orchestration language, which enables you to assemble discrete services into an end-to-end process flow.

BPEL PM

Abbreviation for Business Process Execution Language Process Manager, a comprehensive infrastructure for creating, deploying, and managing BPEL business processes.

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.

build engineer

An actor that is responsible for building, mastering, and packaging artifacts. Some build engineers are responsible for building application artifacts, and some are responsible for building foundation artifacts.

Build Program

A WIN32 executable that reads build configuration files and generates an ANT script for building published business services.

business analyst

An actor that determines if and why an EnterpriseOne business service needs to be developed.

business function

A named set of user-created, reusable business rules and logs that can be called through event rules. Business functions can run a transaction or a subset of a transaction (check inventory, issue work orders, and so on). Business functions also contain the application programming interfaces (APIs) that enable them to be called from a form, a database trigger, or a non-JD Edwards EnterpriseOne application. Business functions can be combined with other business functions, forms, event rules, and other components to make up an application. Business functions can be created through event rules or third-generation languages, such as C. Examples of business functions include Credit Check and Item Availability.

business function event rule

See named event rule (NER).

business service

EnterpriseOne business logic written in Java. A business service is a collection of one or more artifacts. Unless specified otherwise, a business service implies both a published business service and business service.

business service artifacts

Source files, descriptors, and so on that are managed for business service development and are needed for the business service build process.

business service class method

A method that accesses resources provided by the business service framework.

business service configuration files

Configuration files include, but are not limited to, interop.ini, JDBj.ini, and jdelog.properties.

business service cross reference

A key and value data pair used during orchestration. Collectively refers to both the code and the key cross reference in the WSG/XPI based system.

business service cross-reference utilities

Utility services installed in a BPEL/ESB environment that are used to access JD Edwards EnterpriseOne orchestration cross-reference data.

business service development environment

A framework needed by an integration developer to develop and manage business services.

business services development tool

Otherwise known as JDeveloper.

business service EnterpriseOne object

A collection of artifacts managed by EnterpriseOne LCM tools. Named and represented within EnterpriseOne LCM similarly to other EnterpriseOne objects like tables, views, forms, and so on.

business service framework

Parts of the business service foundation that are specifically for supporting business service development.

business service payload

An object that is passed between an enterprise server and a business services server. The business service payload contains the input to the business service when passed to the business services server. The business service payload contains the results from the business service when passed to the Enterprise Server. In the case of notifications, the return business service payload contains the acknowledgement.

business service property

Key value data pairs used to control the behavior or functionality of business services.

Business Service Property Admin Tool

An EnterpriseOne application for developers and administrators to manage business service property records.

business service property business service group

A classification for business service property at the business service level. This is generally a business service name. A business service level contains one or more business service property groups. Each business service property group may contain zero or more business service property records.

business service property key

A unique name that identifies the business service property globally in the system.

business service property utilities

A utility API used in business service development to access EnterpriseOne business service property data.

business service property value

A value for a business service property.

business service repository

A source management system, for example ClearCase, where business service artifacts and build files are stored. Or, a physical directory in network.

business services server

The physical machine where the business services are located. Business services are run on an application server instance.

business services source file or business service class

One type of business service artifact. A text file with the .java file type written to be compiled by a Java compiler.

business service value object template

The structural representation of a business service value object used in a C-business function.

Business Service Value Object Template Utility

A utility used to create a business service value object template from a business service value object.

business services server artifact

The object to be deployed to the business services server.

business view

A means for selecting specific columns from one or more JD Edwards EnterpriseOne application tables whose data is used in an application or report. A business view does not select specific rows, nor does it contain any actual data. It is strictly a view through which you can manipulate data.

central objects merge

A process that blends a customer's modifications to the objects in a current release with objects in a new release.

central server

A server that has been designated to contain the originally installed version of the software (central objects) for deployment to client computers. In a typical JD Edwards EnterpriseOne installation, the software is loaded on to one machine—the central

server. Then, copies of the software are pushed out or downloaded to various workstations attached to it. That way, if the software is altered or corrupted through its use on workstations, an original set of objects (central objects) is always available on the central server.

charts

Tables of information in JD Edwards EnterpriseOne that appear on forms in the software.

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).

checksum

A fixed-size datum computed from an arbitrary block of digital data for the purpose of detecting accidental errors that may have been introduced during its transmission or storage. JD Edwards EnterpriseOne uses the checksum to verify the integrity of packages that have been downloaded by recomputing the checksum of the downloaded package and comparing it with the checksum of the original package. The procedure that yields the checksum from the data is called a checksum function or checksum algorithm. JD Edwards EnterpriseOne uses the MD5 and STA-1 checksum algorithms.

connector

Component-based interoperability model that enables third-party applications and JD Edwards EnterpriseOne to share logic and data. The JD Edwards EnterpriseOne connector architecture includes Java and COM connectors.

Control Table Workbench

An application that, during the Installation Workbench processing, runs the batch applications for the planned merges that update the data dictionary, user-defined codes, menus, and user override tables.

control tables merge

A process that blends a customer's modifications to the control tables with the data that accompanies a new release.

correlation data

The data used to tie HTTP responses with requests that consist of business service name and method.

credentials

A valid set of JD Edwards EnterpriseOne username/password/environment/role, EnterpriseOne session, or EnterpriseOne token.

cross-reference utility services

Utility services installed in a BPEL/ESB environment that access EnterpriseOne cross-reference data.

database credentials

A valid database username/password.

database server

A server in a local area network that maintains a database and performs searches for client computers.

Data Source Workbench

An application that, during the Installation Workbench process, copies all data sources that are defined in the installation plan from the Data Source Master and Table and Data Source Sizing tables in the Planner data source to the system-release number data source. It also updates the Data Source Plan detail record to reflect completion.

deployment artifacts

Artifacts that are needed for the deployment process, such as servers, ports, and such.

deployment server

A server that is used to install, maintain, and distribute software to one or more enterprise servers and client workstations.

direct connect

A transaction method in which a client application communicates interactively and directly with a server application.

See also batch-of-one and store-and-forward.

Do Not Translate (DNT)

A type of data source that must exist on the iSeries because of BLOB restrictions.

embedded application server instance

An OC4J instance started by and running wholly within JDeveloper.

edit code

A code that indicates how a specific value for a report or a form should appear or be formatted. The default edit codes that pertain to reporting require particular attention because they account for a substantial amount of information.

edit mode

A condition of a form that enables users to change data.

edit rule

A method used for formatting and validating user entries against a predefined rule or set of rules.

Electronic Data Interchange (EDI)

An interoperability model that enables paperless computer-to-computer exchange of business transactions between JD Edwards EnterpriseOne and third-party systems. Companies that use EDI must have translator software to convert data from the EDI standard format to the formats of their computer systems.

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.

Employee Work Center

A central location for sending and receiving all JD Edwards EnterpriseOne messages (system and user generated), regardless of the originating application or user. Each user has a mailbox that contains workflow and other messages, including Active Messages.

enterprise server

A server that contains the database and the logic for JD Edwards EnterpriseOne.

Enterprise Service Bus (ESB)

Middleware infrastructure products or technologies based on web services standards that enable a service-oriented architecture using an event-driven and XML-based messaging framework (the bus).

EnterpriseOne administrator

An actor responsible for the EnterpriseOne administration system.

EnterpriseOne credentials

A user ID, password, environment, and role used to validate a user of EnterpriseOne.

EnterpriseOne development client

Historically called "fat client," a collection of installed EnterpriseOne components required to develop EnterpriseOne artifacts, including the Microsoft Windows client and design tools.

EnterpriseOne extension

A JDeveloper component (plug-in) specific to EnterpriseOne. A JDeveloper wizard

is a specific example of an extension.

EnterpriseOne object

A reusable piece of code that is used to build applications. Object types include tables, forms, business functions, data dictionary items, batch processes, business views, event rules, versions, data structures, and media objects.

EnterpriseOne process

A software process that enables JD Edwards EnterpriseOne clients and servers to handle processing requests and run transactions. A client runs one process, and servers can have multiple instances of a process. JD Edwards EnterpriseOne processes can also be dedicated to specific tasks (for example, workflow messages and data replication) to ensure that critical processes don't have to wait if the server is particularly busy.

EnterpriseOne resource

Any EnterpriseOne table, metadata, business function, dictionary information, or other information restricted to authorized users.

Environment Workbench

An application that, during the Installation Workbench process, copies the environment information and Object Configuration Manager tables for each environment from the Planner data source to the system-release number data source. It also updates the Environment Plan detail record to reflect completion.

escalation monitor

A batch process that monitors pending requests or activities and restarts or forwards them to the next step or user after they have been inactive for a specified amount of time.

event rule

A logic statement that instructs the system to perform one or more operations based on an activity that can occur in a specific application, such as entering a form or exiting a field.

explicit transaction

Transaction used by a business service developer to explicitly control the type (auto or manual) and the scope of transaction boundaries within a business service.

exposed method or value object

Published business service source files or parts of published business service source files that are part of the published interface. These are part of the contract with the customer.

fast path

A command prompt that enables the user to move quickly among menus and applications by using specific commands.

file server

A server that stores files to be accessed by other computers on the network. Unlike a disk server, which appears to the user as a remote disk drive, a file server is a sophisticated device that not only stores files, but also manages them and maintains order as network users request files and make changes to these files.

final mode

The report processing mode of a processing mode of a program that updates or creates data records.

foundation

A framework that must be accessible for execution of business services at runtime. This includes, but is not limited to, the Java Connector and JDBj.

FTP server

A server that responds to requests for files via file transfer protocol.

HTTP Adapter

A generic set of services that are used to do the basic HTTP operations, such as GET, POST, PUT, DELETE, TRACE, HEAD, and OPTIONS with the provided URL.

instantiate

A Java term meaning “to create.” When a class is instantiated, a new instance

is created.

integration developer

The user of the system who develops, runs, and debugs the EnterpriseOne business services. The integration developer uses the EnterpriseOne business services to develop these components.

integration point (IP)

The business logic in previous implementations of EnterpriseOne that exposes a document level interface. This type of logic used to be called XBPs. In EnterpriseOne 8.11, IPs are implemented in Web Services Gateway powered by webMethods.

integration server

A server that facilitates interaction between diverse operating systems and applications across internal and external networked computer systems.

integrity test

A process used to supplement a company's internal balancing procedures by locating and reporting balancing problems and data inconsistencies.

interface table

See Z table.

internal method or value object

Business service source files or parts of business service source files that are not part of the published interface. These could be private or protected methods. These could be value objects not used in published methods.

interoperability model

A method for third-party systems to connect to or access JD Edwards EnterpriseOne.

in-your-face error

In JD Edwards EnterpriseOne, a form-level property which, when enabled, causes the text of application errors to appear on the form.

jargon

An alternative data dictionary item description that JD Edwards EnterpriseOne appears based on the product code of the current object.

Java application server

A component-based server that resides in the middle-tier of a server-centric architecture. This server provides middleware services for security and state maintenance, along with data access and persistence.

JDBNET

A database driver that enables heterogeneous servers to access each other's data.

JDEBASE Database Middleware

A JD Edwards EnterpriseOne proprietary database middleware package that provides platform-independent APIs, along with client-to-server access.

JDECallObject

An API used by business functions to invoke other business functions.

jde.ini

A JD Edwards EnterpriseOne file (or member for iSeries) that provides the runtime settings required for JD Edwards EnterpriseOne initialization. Specific versions of the file or member must reside on every machine running JD Edwards EnterpriseOne. This includes workstations and servers.

JDEIPC

Communications programming tools used by server code to regulate access to the same data in multiprocess environments, communicate and coordinate between processes, and create new processes.

jde.log

The main diagnostic log file of JD Edwards EnterpriseOne. This file is always located in the root directory on the primary drive and contains status and error messages from the startup and operation of JD Edwards EnterpriseOne.

JDENET

A JD Edwards EnterpriseOne proprietary communications middleware package. This package is a peer-to-peer, message-based, socket-based, multiprocess communications middleware solution. It handles client-to-server and server-to-server communications for all JD Edwards EnterpriseOne supported platforms.

JDeveloper Project

An artifact that JDeveloper uses to categorize and compile source files.

JDeveloper Workspace

An artifact that JDeveloper uses to organize project files. It contains one or more project files.

JMS Queue

A Java Messaging service queue used for point-to-point messaging.

listener service

A listener that listens for XML messages over HTTP.

local repository

A developer's local development environment that is used to store business service artifacts.

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.

logic server

A server in a distributed network that provides the business logic for an application program. In a typical configuration, pristine objects are replicated on to the logic server from the central server. The logic server, in conjunction with workstations, actually performs the processing required when JD Edwards EnterpriseOne software runs.

MailMerge Workbench

An application that merges Microsoft Word 6.0 (or higher) word-processing documents with JD Edwards EnterpriseOne records to automatically print business documents. You can use MailMerge Workbench to print documents, such as form letters about verification of employment.

Manual Commit transaction

A database connection where all database operations delay writing to the database until a call to commit is made.

master business function (MBF)

An interactive master file that serves as a central location for adding, changing, and updating information in a database. Master business functions pass information between data entry forms and the appropriate tables. These master functions provide a common set of functions that contain all of the necessary default and editing rules for related programs. MBFs contain logic that ensures the integrity of adding, updating, and deleting information from databases.

master table

See published table.

media storage object

Files that use one of the following naming conventions that are not organized into table format: Gxxx, xxxGT, or GTxxx.

message center

A central location for sending and receiving all JD Edwards EnterpriseOne messages (system and user generated), regardless of the originating application or user.

messaging adapter

An interoperability model that enables third-party systems to connect to JD Edwards EnterpriseOne to exchange information through the use of messaging queues.

messaging server

A server that handles messages that are sent for use by other programs using a messaging API. Messaging servers typically employ a middleware program to perform their functions.

Monitoring Application

An EnterpriseOne tool provided for an administrator to get statistical information for various EnterpriseOne servers, reset statistics, and set notifications.

named event rule (NER)

Encapsulated, reusable business logic created using event rules, rather that C programming. NERs are also called business function event rules. NERs can be reused in multiple places by multiple programs. This modularity lends itself to streamlining, reusability of code, and less work.

Object Configuration Manager (OCM)

In JD Edwards EnterpriseOne, the object request broker and control center for the runtime environment. OCM keeps track of the runtime locations for business functions, data, and batch applications. When one of these objects is called, OCM directs access to it using defaults and overrides for a given environment and user.

Object Librarian

A repository of all versions, applications, and business functions reusable in building applications. Object Librarian provides check-out and check-incapabilities for developers, and it controls the creation, modification, and use of JD Edwards EnterpriseOne objects. Object Librarian supports multiple environments (such as

production and development) and enables objects to be easily moved from one environment to another.

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.

Open Data Access (ODA)

An interoperability model that enables you to use SQL statements to extract JD Edwards EnterpriseOne data for summarization and report generation.

Output Stream Access (OSA)

An interoperability model that enables you to set up an interface for JD Edwards EnterpriseOne to pass data to another software package, such as Microsoft Excel, for processing.

package

JD Edwards EnterpriseOne objects are installed to workstations in packages from the deployment server. A package can be compared to a bill of material or kit that indicates the necessary objects for that workstation and where on the deployment server the installation program can find them. It is point-in-time snapshot of the central objects on the deployment server.

package build

A software application that facilitates the deployment of software changes and new applications to existing users. Additionally, in JD Edwards EnterpriseOne, a package build can be a compiled version of the software. When you upgrade your version of the ERP software, for example, you are said to take a package build.

Consider the following context: "Also, do not transfer business functions into the production path code until you are ready to deploy, because a global build of business functions done during a package build will automatically include the new functions." The process of creating a package build is often referred to, as it is in this example, simply as "a package build."

package location

The directory structure location for the package and its set of replicated objects. This is usually \\deployment server\release\path_code\package\package name. The subdirectories under this path are where the replicated objects for the package are placed. This is also referred to as where the package is built or stored.

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.

Pathcode Directory

The specific portion of the file system on the EnterpriseOne development client where EnterpriseOne development artifacts are stored.

patterns

General repeatable solutions to a commonly occurring problem in software design. For business service development, the focus is on the object relationships and interactions.

For orchestrations, the focus is on the integration patterns (for example, synchronous and asynchronous request/response, publish, notify, and receive/reply).

print server

The interface between a printer and a network that enables network clients to connect to the printer and send their print jobs to it. A print server can be a computer, separate hardware device, or even hardware that resides inside of the printer itself.

pristine environment

A JD Edwards EnterpriseOne environment used to test unaltered objects with JD Edwards EnterpriseOne demonstration data or for training classes. You must have this environment so that you can compare pristine objects that you modify.

processing option

A data structure that enables users to supply parameters that regulate the running of a batch program or report. For example, you can use processing options to specify default values for certain fields, to determine how information appears or is printed, to specify date ranges, to supply runtime values that regulate program execution, and so on.

production environment

A JD Edwards EnterpriseOne environment in which users operate EnterpriseOne software.

Production Published Business Services Web Service

Published business services web service deployed to a production application server.

program temporary fix (PTF)

A representation of changes to JD Edwards EnterpriseOne software that your organization receives on magnetic tapes or disks.

project

In JD Edwards EnterpriseOne, a virtual container for objects being developed in Object Management Workbench.

promotion path

The designated path for advancing objects or projects in a workflow. The following is the normal promotion cycle (path):

11>21>26>28>38>01

In this path, 11 equals new project pending review, 21 equals programming, 26 equals QA test/review, 28 equals QA test/review complete, 38 equals in production, 01 equals complete. During the normal project promotion cycle, developers check objects out of and into the development path code and then promote them to the prototype path code. The objects are then moved to the productions path code before declaring them complete.

proxy server

A server that acts as a barrier between a workstation and the internet so that the enterprise can ensure security, administrative control, and caching service.

published business service

EnterpriseOne service level logic and interface. A classification of a published business service indicating the intention to be exposed to external (non-EnterpriseOne) systems.

published business service identification information

Information about a published business service used to determine relevant authorization records. Published business services + method name, published business services, or *ALL.

published business service web service

Published business services components packaged as J2EE Web Service (namely, a J2EE EAR file that contains business service classes, business service foundation, configuration files, and web service artifacts).

published table

Also called a master table, this is the central copy to be replicated to other machines. Residing on the publisher machine, the F98DRPUB table identifies all of the published tables and their associated publishers in the enterprise.

publisher

The server that is responsible for the published table. The F98DRPUB table identifies all of the published tables and their associated publishers in the enterprise.

QBE

An abbreviation for query by example. In JD Edwards EnterpriseOne, the QBE line is the top line on a detail area that is used for filtering data.

real-time event

A message triggered from EnterpriseOne application logic that is intended for external systems to consume.

refresh

A function used to modify JD Edwards EnterpriseOne software, or subset of it, such as a table or business data, so that it functions at a new release or cumulative update level.

replication server

A server that is responsible for replicating central objects to client machines.

rules

Mandatory guidelines that are not enforced by tooling, but must be followed in order to accomplish the desired results and to meet specified standards.

secure by default

A security model that assumes that a user does not have permission to execute an object unless there is a specific record indicating such permissions.

Secure Socket Layer (SSL)

A security protocol that provides communication privacy. SSL enables client and server applications to communicate in a way that is designed to prevent eavesdropping, tampering, and message forgery.

selection

Found on JD Edwards EnterpriseOne menus, a selection represents functions that you can access from a menu. To make a selection, type the associated number in the Selection field and press Enter.

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.

Server Workbench

An application that, during the Installation Workbench process, copies the server configuration files from the Planner data source to the system-release number data source. The application also updates the Server Plan detail record to reflect completion.

SOA

Abbreviation for Service Oriented Architecture.

softcoding

A coding technique that enables an administrator to manipulate site-specific variables that affect the execution of a given process.

source repository

A repository for HTTP adapter and listener service development environment artifacts.

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.

specification

A complete description of a JD Edwards EnterpriseOne object. Each object has its own specification, or name, which is used to build applications.

Specification Table Merge Workbench

An application that, during the Installation Workbench process, runs the batch applications that update the specification tables.

SSL Certificate

A special message signed by a certificate authority that contains the name of a user and that user's public key in such a way that anyone can "verify" that the message was signed by no one other than the certification authority and thereby develop trust in the user's public key.

store-and-forward

The mode of processing that enables users who are disconnected from a server to enter transactions and then later connect to the server to upload those transactions.

subscriber table

Table F98DRSUB, which is stored on the publisher server with the F98DRPUB table and identifies all of the subscriber machines for each published table.

super class

An inheritance concept of the Java language where a class is an instance of something, but is also more specific. “Tree” might be the super class of “Oak” and “Elm,” for example.

table access management (TAM)

The JD Edwards EnterpriseOne component that handles the storage and retrieval of use-defined data. TAM stores information, such as data dictionary definitions; application and report specifications; event rules; table definitions; business function input parameters and library information; and data structure definitions for running applications, reports, and business functions.

Table Conversion Workbench

An interoperability model that enables the exchange of information between JD Edwards EnterpriseOne and third-party systems using non-JD Edwards EnterpriseOne tables.

table conversion

An interoperability model that enables the exchange of information between JD Edwards EnterpriseOne and third-party systems using non-JD Edwards EnterpriseOne tables.

table event rules

Logic that is attached to database triggers that runs whenever the action specified by the trigger occurs against the table. Although JD Edwards EnterpriseOne enables event rules to be attached to application events, this functionality is application specific. Table event rules provide embedded logic at the table level.

terminal server

A server that enables terminals, microcomputers, and other devices to connect to a network or host computer or to devices attached to that particular computer.

transaction processing (TP) monitor

A monitor that controls data transfer between local and remote terminals and the applications that originated them. TP monitors also protect data integrity in the distributed environment and may include programs that validate data and format terminal screens.

transaction processing method

A method related to the management of a manual commit transaction boundary (for example, start, commit, rollback, and cancel).

transaction set

An electronic business transaction (electronic data interchange standard document) made up of segments.

trigger

One of several events specific to data dictionary items. You can attach logic to a data dictionary item that the system processes automatically when the event occurs.

triggering event

A specific workflow event that requires special action or has defined consequences or resulting actions.

user identification information

User ID, role, or *public.

User Overrides merge

Adds new user override records into a customer's user override table.

value object

A specific type of source file that holds input or output data, much like a data structure passes data. Value objects can be exposed (used in a published business service) or internal, and input or output. They are comprised of simple and complex elements and accessories to those elements.

versioning a published business service

Adding additional functionality/interfaces to the published business services without modifying the existing functionality/interfaces.

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.

visual assist

Forms that can be invoked from a control via a trigger to assist the user in determining what data belongs in the control.

vocabulary override

An alternate description for a data dictionary item that appears on a specific JD Edwards EnterpriseOne form or report.

web application server

A web server that enables web applications to exchange data with the back-end systems and databases used in eBusiness transactions.

web server

A server that sends information as requested by a browser, using the TCP/IP set of protocols. A web server can do more than just coordination of requests from browsers; it can do anything a normal server can do, such as house applications or data. Any computer can be turned into a web server by installing server software and connecting the machine to the internet.

Web Service Description Language (WSDL)

An XML format for describing network services.

Web Service Inspection Language (WSIL)

An XML format for assisting in the inspection of a site for available services and a set of rules for how inspection-related information should be made.

web service softcoding record

An XML document that contains values that are used to configure a web service proxy. This document identifies the endpoint and conditionally includes security information.

web service softcoding template

An XML document that provides the structure for a soft coded record.

Where clause

The portion of a database operation that specifies which records the database operation will affect.

Windows terminal server

A multiuser server that enables terminals and minimally configured computers to display Windows applications even if they are not capable of running Windows software themselves. All client processing is performed centrally at the Windows terminal server and only display, keystroke, and mouse commands are transmitted over the network to the client terminal device.

wizard

A type of JDeveloper extension used to walk the user through a series of steps.

workbench

A program that enables users to access a group of related programs from a single entry point. Typically, the programs that you access from a workbench are used to complete a large business process. For example, you use the JD Edwards EnterpriseOne Payroll Cycle Workbench (P07210) to access all of the programs that the system uses to process payroll, print payments, create payroll reports, create journal entries, and update payroll history. Examples of JD Edwards EnterpriseOne workbenches include Service Management Workbench (P90CD020), Line Scheduling Workbench (P3153), Planning Workbench (P13700), Auditor's Workbench (P09E115), and Payroll Cycle Workbench.

workflow

The automation of a business process, in whole or in part, during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules.

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.

XAPI events

A service that uses system calls to capture JD Edwards EnterpriseOne transactions as they occur and then calls third-party software, end users, and other JD Edwards EnterpriseOne systems that have requested notification when the specified transactions occur to return a response.

XML CallObject

An interoperability capability that enables you to call business functions.

XML Dispatch

An interoperability capability that provides a single point of entry for all XML documents coming into JD Edwards EnterpriseOne for responses.

XML List

An interoperability capability that enables you to request and receive JD Edwards EnterpriseOne database information in chunks.

XML Service

An interoperability capability that enables you to request events from one JD Edwards EnterpriseOne system and receive a response from another JD Edwards EnterpriseOne system.

XML Transaction

An interoperability capability that enables you to use a predefined transaction type to send information to or request information from JD Edwards EnterpriseOne. XML transaction uses interface table functionality.

XML Transaction Service (XTS)

Transforms an XML document that is not in the JD Edwards EnterpriseOne format into an XML document that can be processed by JD Edwards EnterpriseOne. XTS then transforms the response back to the request originator XML format.

Z event

A service that uses interface table functionality to capture JD Edwards EnterpriseOne transactions and provide notification to third-party software, end users, and other JD Edwards EnterpriseOne systems that have requested to be notified when certain transactions occur.

Z table

A working table where non-JD Edwards EnterpriseOne information can be stored and then processed into JD Edwards EnterpriseOne. Z tables also can be used to retrieve JD Edwards EnterpriseOne data. Z tables are also known as interface tables.

Z transaction

Third-party data that is properly formatted in interface tables for updating to the JD Edwards EnterpriseOne database.

Index

J

K

L

M

N

O

P

R

S

T

U

W