

**OpenText™ Exstream™**

## **Preparing Applications for Production**

Design and Production Documentation

Release 16.6.0

**OpenText™ Exstream**  
**Preparing Applications for Production**  
Rev.: 2019-Apr-30

**This documentation has been created for software version 16.6.0.**

It is also valid for subsequent software versions as long as no new document version is shipped with the product or is published at <https://knowledge.opentext.com>.

**Open Text Corporation**  
275 Frank Tompa Drive, Waterloo, Ontario, Canada, N2L 0A1  
Tel: +1-519-888-7111  
Toll Free Canada/USA: 1-800-499-6544 International: +800-4996-5440  
Fax: +1-519-888-0677

Support: <https://support.opentext.com>  
For more information, visit <https://www.opentext.com>

**Copyright © 2019 Open Text. All rights reserved.**

Trademarks owned by Open Text.

One or more patents may cover this product. For more information, please visit, <https://www.opentext.com/patents>

#### **Disclaimer**

##### **No Warranties and Limitation of Liability**

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However, Open Text Corporation and its affiliates accept no responsibility and offer no warranty whether expressed or implied, for the accuracy of this publication.

# Contents

Chapter 1: Application Development in Exstream .....	6
1.1 Using the Design Environment to Package, Test, and Troubleshoot Your Application .....	6
1.1.1 Targeting the Implementation That Best Fits Your Quality Assurance Goals ..	7
1.2 Using the Production Environment to Customize and Control Engine Processes ..	8
1.2.1 Targeting the Implementation That Best Fits Your Document Production Goals .....	9
1.3 Completing the Packaging Process .....	9
Chapter 2: Testing Applications Using the Design Engine .....	10
2.1 Setting Engine Options .....	11
2.1.1 Customizing Engine Behavior .....	12
2.1.2 Customizing How Design Manager Issues Informational Message Files .....	14
2.1.3 Configuring Settings for Message and Report Files .....	16
2.1.4 Tracking Application Development to Better Manage Campaigns .....	18
2.2 Setting Options for Testing Tools .....	19
2.2.1 Locating and Identifying Errors in an Application .....	19
2.2.2 Generating a Rule Analysis Report to Validate Application Rules .....	20
2.2.3 Capturing Test Data to Collect and identify Rules and Customers in Each Statement .....	21
2.3 Executing a Test Engine Run .....	22
2.4 Validating Application Changes .....	24
2.4.1 Viewing and Comparing Applications in Designer .....	26
2.4.2 Viewing AFP or Metacode Compare Files for Testing and Comparison .....	29
2.5 Viewing Package Reports .....	30
Chapter 3: Packaging an Application .....	32
3.1 About Packaging an Application .....	32
3.2 Packaging Applications in Design Manager .....	34
3.2.1 Creating and Customizing a Package File .....	34
3.2.2 Uploading a package file to the CAS repository .....	41
3.2.3 Managing Object Versions in Your Package File .....	44
3.2.4 Specifying Resource Packaging Options .....	46
3.2.5 Managing Package Profiles .....	48

3.2.6	Packaging for Previews .....	49
3.3	Packaging Applications From the Command Prompt .....	53
3.3.1	Performing Multiple Packaging Operations From the Command Line .....	55
3.3.2	Using Packaging Switches to Create Campaign and Document Package Files .....	55
3.3.3	Uploading a package file to the CAS repository from the command prompt ..	56
3.3.4	Packaging switches .....	59
3.4	Increasing Productivity with a Reusable Packaging Format .....	71
3.5	Integrating Changes at the Application Level .....	72
3.5.1	Creating Sub-Packages for Campaigns, Documents, and Objects .....	73
3.6	Optimizing the Packaging Process Through Consolidating Applications .....	75
3.7	Implementing Collaborative Authoring Within the Packaging Process .....	75
3.7.1	Combining Files from Multiple Databases to Integrate Collaborative Authoring .....	76
3.8	Managing Application Objects With Version Labels .....	76
3.8.1	Creating A New Version Label .....	77
3.8.2	Viewing the Objects in a Labeled Application .....	79
3.8.3	Selecting a Version Label When Packaging an Application .....	80
<b>Chapter 4:</b>	<b>Building a Control File .....</b>	<b>81</b>
4.1	Writing a Control File to Manage Application Production .....	81
4.1.1	Writing Control Files for the z/OS Production Engine .....	82
4.2	Validating Your License Key in the Control File to Avoid Engine Errors .....	84
4.2.1	Inserting the License Key on Windows and UNIX Platforms .....	84
4.2.2	Inserting the License Key on the z/OS Platform .....	85
4.3	Specifying Multiple Package File Operations in the Control File .....	86
4.4	Executing a Production Engine Run with a Control File .....	86
<b>Chapter 5:</b>	<b>Optimizing Production Engine Performance .....</b>	<b>88</b>
5.1	Maximizing Engine Processing Capabilities .....	88
5.1.1	Understanding Application Packaging Concepts to Achieve Optimal Engine Performance .....	89
5.1.2	Understanding Resource Management Concepts to Achieve Optimal Engine Performance .....	90
5.1.3	Addressing Packaging Messages to Analyze Engine Results .....	90
5.1.4	How the Engine Uses the Package File to Determine Special Features and Allocate Memory .....	91
5.2	Understanding Engine Processing and Timing to Improve Production .....	91

5.2.1 Engine Processing with General Data and Data Sections .....	92
5.2.2 Post-Sort Engine Processing with General Data and Data Sections .....	99
5.2.3 Engine Processing with Schema Model Data Files .....	103
5.2.4 Post-Sort Engine Processing with Schema Model Data Files .....	109
5.3 Managing Production Engine Processing Memory .....	113
5.3.1 Resetting Memory with the Start of Each New Section/Customer .....	114
5.3.2 Writing Table Row Data to Specified Temporary Files .....	115
5.3.3 Applying Memory Management Techniques to Maximize Engine Potential for the Production Engine on z/OS .....	116
<b>Chapter 6: Running the Production Engine .....</b>	<b>123</b>
6.1 Preparing the Application for Production Engine Processing .....	123
6.1.1 Retaining Naming Conventions for Directory and Production File Names ...	124
6.1.2 Transferring Package Files from the Design Environment to the Production Environment .....	124
6.2 Using the Command Prompt to Execute a Production Engine Run .....	125
6.2.1 Executing an Engine Run .....	126
6.2.2 Executing an Engine Run From a UNIX/Linux Shell Prompt or Shell Script .	126
6.2.3 Executing an Engine Run From an AS/400 Shell Environment .....	128
6.2.4 Executing an Engine Run on z/OS Command Line Interface .....	128
6.3 Special Considerations for Running Multiple Instances of the Engine .....	137
6.4 Understanding System Generated Files for the Production Engine .....	137
6.4.1 Generating Message Files While Processing Applications .....	137
6.4.2 Generating Report Files to Show Details of Each Application Object .....	138
6.5 Comparing and Viewing Exstream Batch Compare Files .....	143
6.5.1 Viewing and Comparing Applications Using Exstream Batch Compare .....	145
6.6 Engine Return Codes .....	147

# Chapter 1: Application Development in Exstream

Much like designing in Exstream, developing an application is the result of the consolidated effort of multiple people in your organization. An application consists of all the required elements for document creation, except the file format you use to send your documents through the engine. The point at which you begin developing an application occurs after all other preparation and design steps are complete. When the application is complete, you must then package the application(s) into a format that is optimized for the Exstream production engine.

This guide focuses on the concepts and tasks associated with developing an application, testing and troubleshooting the application, and preparing the application for the final stages of production. With the information this guide provides, you can develop a set of implementations that appeal to the specific business processes you require to complete customer document communications and fulfillment.

This chapter discusses the following topics:

- [“Using the Design Environment to Package, Test, and Troubleshoot Your Application” below](#)
- [“Using the Production Environment to Customize and Control Engine Processes” on page 8](#)
- [“Completing the Packaging Process” on page 9](#)

## 1.1 Using the Design Environment to Package, Test, and Troubleshoot Your Application

The first phase in completing the development of your application occurs in the design environment where you can use Designer and Design Manager to package, test, and troubleshoot your application. In the application development process, you want to focus on several features and tools that allow you to ensure the quality and accuracy of your documents. You use features in Design Manager to perform testing and troubleshooting tasks. In the testing and troubleshooting phases, you use Designer as a comparison tool, which lets you visually compare existing applications with updated or new applications.

Using the testing tools available in Exstream helps you avoid potential errors at engine run time, which is essential in improving your organization's production workflow.

The design environment provides the following testing and troubleshooting features:

- **Debug/Trace**—The Debug/Trace feature utilizes two main tools: a trace capability and a watch on key objects, which allows you to locate and identify potential errors in your application.
- **Rule Analyzer**—The Rule Analyzer module lets you test how rules within an application are executed.
- **Test Data Capture**—The Test Data Capture module lets you collect a minimal amount of test data to fully exercises all named and unnamed rules in an application.

For more information about testing and troubleshooting features, see [“Testing Applications Using the Design Engine” on page 10](#).

## 1.1.1 Targeting the Implementation That Best Fits Your Quality Assurance Goals

Because the processes and best practices of each organization are different, it is important to target an implementation that meets the needs of your quality assurance goals. Typically, a system administrator or quality assurance representative handles the tasks associated with initial packaging and testing of the application before it is sent to the production stages. How your company decides to assign user roles and processes is up to the system administrator, but typical implementations of Design Manager application development processes can range from a single user performing all tasks from design through testing, to multiple designers contributing to the application. If multiple designers contribute to the application, it might be easiest to assign a system administrator to handle the database and a primary designer to be responsible for collecting all of the design elements and ensuring the application is ready for publication.

When the system administrator (or the design user in charge of finished applications) verifies the status of the application, testing must be done to ensure that all design elements included in the application do not interfere with the performance of the engine. In many cases, it is also necessary to use the comparison tools available in Design Manager. The comparison tools let a user compare two or more applications to ensure that new or added design elements do not affect other design objects in the application. Typically, someone in a quality assurance role is in charge of testing and comparing applications before they move to the production stages.

For example, suppose you work in the quality assurance department of a large financial institution and your company wants to update the design of old documents using Exstream technology. The system administrator sends you a new application from the design team and you must first test the application using the Design Manager interface. Using testing tools, such as the Rule Analyzer and Test Data Capture modules, you ensure the application is prepared and does not cause performance issues when it is run through the production engine.

For more information about testing and comparing applications in Design Manager, see [“Testing Applications Using the Design Engine” on page 10](#)

## 1.2 Using the Production Environment to Customize and Control Engine Processes

After you target the implementation for testing and comparing applications in the design environment, the next phase in completing the development of your application occurs in the production environment where you can customize and control how the engine processes and produces your applications. The following features in the production environment give you the flexibility and leverage to meet the growing demands and needs of both your organization and its customers:

- Reusable control files—Control files are single, reusable files that centralize engine switches required to run the engine, which help your company reduce production costs and time.
- A 64-bit Exstream production engine—As more applications become transaction-intensive (especially in financial communications) memory can become an issue. The 64-bit engine availability offers your organization a simple way to avoid memory management issues during production.

A 32-bit production engine is also available. You should verify whether the platform server on which you want to install the production engine is 32-bit or 64-bit. For example, to install the 64-bit version of the Exstream production system, you must make sure you use 64-bit capable hardware with a 64-bit operating system.

For more information about the 32-bit and 64-bit production engines, see [“Optimizing Production Engine Performance” on page 88](#).

The production environment is divided into two parts: the production engine and output. When you prepare to send a package file through the engine to produce output, it is best to ensure all design and testing is complete. In most business environments, the engine is located on a separate operating system, such as UNIX or z/OS. Offering the production environment on multiple platforms gives your company more leverage to integrate the Exstream platform into your existing infrastructure. For example, suppose your company's design team operates on a Windows platform and your IT department runs the technological infrastructure from a UNIX platform. With the Exstream platform, the UNIX version of the production engine fits into the IT department's existing infrastructure.

In the production environment, the final stages of the packaging process relates only to the production engine. Typically, a developer or IT employee in your organization handles the production engine tasks associated with the final stages of production, such as writing a reusable control file that contains all of the necessary engine switches to run the engine and produce documents.

For more information about output, see *Creating Output* in the Exstream Design and Production documentation.

For more information about control files, see [“Building a Control File” on page 81](#).



## 1.2.1 Targeting the Implementation That Best Fits Your Document Production Goals

Because the processes and best practices of each organization are different, it is important to target an implementation that meets the needs of your document production goals. Typically, a system administrator delivers the package file to a development team, who reviews specifications, writes a control file, and executes an engine run. If no errors occur when the engine runs, designated output queues receive the information and the application is ready for printing.

For example, suppose you are on the production development team for a large phone company. You must perform all production engine tasks to ensure millions of customers receive their documents. After you review the specifications sent by the system administrator, you review the necessary engine switches you need for this production run. After you collect all of the necessary engine switches, you open a standard text editor, such as Windows Notepad, and begin building a control file. You save the reusable control file for additional production runs. After you place the location of the control file on the command line, you execute an engine run. All information is processed and sent to the designated output queues where they can be printed.

For more information about output and output queues, see *Creating Output* in the Exstream Design and Production documentation.

## 1.3 Completing the Packaging Process

Completing the packaging process requires the combination of many elements in the Exstream environment. The application development begins when you combine resources, data files, output devices, and documents into a single application. Next, the application is packaged into a format that is optimized for the engine and the package file is sent through the Exstream engine, which generates the application. Finally, the generated application is sent through an output device and your organization's customer communications are produced.

## Chapter 2: Testing Applications Using the Design Engine

The design engine gives you the ability to test your applications within the confines of the Design Manager interface, or, more specifically, from the **Run the Engine** dialog box. The **Run the Engine** dialog box lets you set all of the same options you would when generating an actual production engine run with the added advantages of comprehensive testing tools. Each of the testing tools available in Exstream is specifically designed to help you achieve your organization's quality assurance goals.

The options on the **Run the Engine** dialog box control how the engine runs, error reporting and tracking, and how the files are created. Talk with your system administrator to ensure the file extensions are correct.

Certain options on the **Run the Engine** dialog box are active only if your company has licensed certain modules or suites. For example, the **Tracking** area is available only if you have licensed the Advanced Campaign Management module. Check with your system administrator to verify which options are available to you.

To test applications in the design environment, complete the following tasks as needed:

- [“Setting Engine Options” on the next page](#)
- [“Setting Options for Testing Tools” on page 19](#)
- [“Executing a Test Engine Run” on page 22](#)
- [“Validating Application Changes” on page 24](#)
- [“Viewing Package Reports” on page 30](#)

### Run the Engine dialog box

**Run the Engine**

Basic options (any options set in the control file will take precedence)

- ☒ Control file: ExstreamControl.dat
- ☒ Package file: ExstreamPackage.pub
- ☒ Compose file: ExstreamOutput.ecf
- ☒ Key file:
- ☒ Customers: 1 to 10 by 1
- ☒ Functions: Compose documents
- ☒ Run date: 2017 March 17
- ☒ Mode: Local
- ☒ Create output: Yes
- ☒ Error stop level: Severe
- ☐ Production output (requires Workstation Engine license)

View message file

When: Ask ☐ Group messages

Which: Information ☐ No repeats

☒ View output when complete  
(if no errors and composing for Exstream viewer or alternate viewer)

Reports and messages

- ☒ Message file: ExstreamMessages.dat
- ☒ XML Message file: ExstreamMessages.xml
- ☒ Message level: All (info, warning, and error)
- ☒ Report file: ExstreamReport.dat
- ☒ Reporting level: Selection summary

Tracking

- ☐ Track results: None
- ☐ Track file:
- ☐ Tracking input: Database

Testing

- ☒ Rule analysis: RuleAnalysisReport.txt
- ☐ Detailed Sort by: Unexecuted Lines
- ☒ Capture test data:
- Customers per rule statement: 1
- Reporting: Summary
- Report: TestDataCaptureReport.txt
- Optimization: Normal

Debug/trace

- ☒ Debug file: ExstreamDebug.txt
- ☐ Trace all
- ☒ Trace logic
  - ☐ Document
  - ☐ Page
  - ☐ Frame
  - ☐ Message
  - ☐ Campaign
- ☐ Trace data
- ☒ Watch level: Variable



Run Cancel

## 2.1 Setting Engine Options

To begin the testing phases of application development, you must first set engine options in the **Run the Engine** dialog box. Four main areas of the dialog box let you control how the design engine behaves when generating applications in the package file:

- **Basic options** area
- **View message file** area
- **Reports and messages** area
- **Tracking** area



Typically, a system administrator will provide you with the appropriate package file that contains the application(s) you must test. After you receive the package file, you must first access the **Run the Engine** dialog box in one of the following ways:



- From the Menu bar, select **Tools > Run Engine**.
- On the **Build Package** dialog box, click .
- On the **Standard** toolbar, click .

### 2.1.1 Customizing Engine Behavior

When you select options under the **Basic options** area, you are customizing engine behavior, including how the engine handles the output file, which customers to include, and when the file is created. Each check box in the **Basic options** area acts as an engine switch. Engine switches control aspects of how the engine behaves. For example, selecting the **Package File** check box has the same effect as entering the PACKAGEFILE engine switch in a control file or from the command prompt to execute a production engine run.

To customize engine behavior, complete the following tasks:

To	Do this
Send instructions to the engine using a control file	<div><div><div>1. Select the <b>Control file</b> check box. The box to the right becomes active.</div><div>2. To browse to the location of the control file, click  or enter the name of the file in the <b>Control file</b> box.</div></div><div>A control file uses engine switches to turn options on and off without using the interface. If you select the <b>Control file</b> check box and the control file contains all the necessary switches, clear the remaining check boxes on the <b>Run the Engine</b> dialog box.</div></div>
Specify the package file location	<div><div><div>1. Select the <b>Package file</b> check box. The box to the right becomes active.</div><div>2. To browse to the location of the control file, click  or enter the name of the file in the name and location of the package file in the <b>Package file</b> box.</div></div><div>You can use the full path or a local file name. If you use a local file name, the default location is as follows: C:\Program Files\OpenText\Exstream\Exstream &lt;version&gt;</div></div>

To	Do this
Create an output file for your output device	<ol style="list-style-type: none"> <li>1. Select the <b>Compose file</b> check box. The box to the right becomes active.</li> <li>2. To browse to the location of the control file, click  or enter the name of the file in the <b>Compose file</b> box. For example: ExstreamOutput.pdf.</li> </ol> <p>The <b>Compose file</b> option is used only when you package with the <b>Specify output</b> option. If output queues are used, the output file defined in the queues is used. Queues are used with the High-Volume Delivery module only.</p> <div style="border: 1px solid black; padding: 5px;"> <p><b>Note:</b> If you are packaging for an output device such as PDF, you might need to add or change the extension of the compose file.</p> </div>
Specify a key file for the engine run	<ol style="list-style-type: none"> <li>1. Select the <b>Key file</b> check box. This overrides the key in the <b>System Settings</b>.</li> <li>2. To browse to the location of the control file, click  or enter the name of the file in the <b>Compose file</b> box.</li> </ol> <p>You can use the full path or a local file name. If you use a local file name, the default location is: C:\Program Files\OpenText\Exstream\Exstream &lt;version&gt;</p>
Limit the run to specific customers	<ol style="list-style-type: none"> <li>1. Select the <b>Customers</b> check box.</li> <li>2. In the first and second boxes to the right, enter the first and last customer to process in the next box.</li> <li>3. In the last box to the right, enter the step amount.</li> </ol> <p>For example, if you enter 1 to 100 by 10, every tenth customer is processed up to customer 100.</p>
Determine what the engine produces using the package file	<ol style="list-style-type: none"> <li>1. Select the <b>Functions</b> check box.</li> <li>2. From the drop-down list, select one of the following: <ul style="list-style-type: none"> <li>• <b>Compose documents</b>—Compose the documents in the package file so they can be printed.</li> <li>• <b>Inserter control</b>—Determine which inserts are available and set inserter system variables.</li> <li>• <b>Unlimited campaign and document selection</b>—Determine which campaigns and documents qualify for an application. This is similar to the application test.</li> <li>• <b>Data dictionary analysis</b>—Determine whether the data file is mapped properly.</li> <li>• <b>Package content report</b>—Generate an report of the current package contents to check that all the necessary components are included.</li> </ul> </li> </ol>
Compose using a specified date	<ol style="list-style-type: none"> <li>1. Select the <b>Run date</b> check box. The current date appears in the boxes to the right.</li> <li>2. Select the month, day, and year from their respective boxes.</li> </ol> <p>The date you specify here is usually a date other than today's date. The objects are processed as they exist or existed on the date you specify.</p>

To	Do this
Determine the data source the engine uses at run time	<ol style="list-style-type: none"> <li>1. Select the <b>Mode</b> check box.</li> <li>2. From the drop-down list, select one of the following: <ul style="list-style-type: none"> <li>• <b>Local</b>—The engine uses the test data source you specified for objects, such as data files and output queues.</li> <li>• <b>Production</b>—The engine uses the production data source you specified for objects, such as data files and output queues.</li> </ul> </li> </ol> <div> <p><b>Note:</b> If your system is not enabled for production, <b>Mode</b> automatically switches to <b>Local</b> if you select <b>Production</b>.</p> </div>
Create output files from the composed data	<ol style="list-style-type: none"> <li>1. Select the <b>Create output</b> check box.</li> <li>2. From the drop-down list to the right of the <b>Create output</b> check box, select <b>Yes</b>. (Select <b>No</b> to run the engine but not produce output).</li> </ol> <p>Selecting <b>No</b> is often used for testing large applications with production data. It is faster and uses less memory space than producing output.</p>
Stop engine processing when a message is issued at or above the specified level	<ol style="list-style-type: none"> <li>1. Select the <b>Error stop level</b> check box.</li> <li>2. From the drop-down list to the right of the <b>Error stop level</b> check box, select one of the following options: <ul style="list-style-type: none"> <li>• <b>Warning</b>—Stops engine processing when Design Manager issues a warning, error, or severe level message</li> <li>• <b>Error</b>—Stops engine processing when Design Manager issues an error or severe level message</li> <li>• <b>Severe</b>—Stops engine processing when Design Manager issues a severe level message</li> </ul> </li> </ol>

## 2.1.2 Customizing How Design Manager Issues Informational Message Files

The **View message file** area lets you customize how Design Manager issues informational message files. Message files show you all of the messages the engine generates when processing an application. Message files are system-generated files that can help you identify and troubleshoot potential problems that can be encountered at engine run time. Customizing the way in which you receive these message files can help you prevent problems when you enter the production stages of application development.

For more information about system-generated files in the production environment, see [“Running the Production Engine” on page 123](#).

To customize how Design Manager issues informational message files:

1. From the **When** drop-down list, select from the following options when you want a Design Manager-issued message window to open:
  - **Ask**— Design Manager issues a message asking if you want to view messages.
  - **If errors**— Design Manager issues messages if any are at the error level.
  - **If warnings**— Design Manager issues messages if any are at the warning level.
  - **Always**— Design Manager always issues messages.
  - **Never**— Design Manager never issues messages.
2. From the **Which** drop-down list, select which level of errors appear:
  - **Information**—Messages appear at the information, warning, error, and severe levels.
  - **Warning**—Messages appear at the warning, error, and severe levels.
  - **Error**—Messages appear at the error and severe levels.
  - **Severe**—Messages appear at the severe levels.
3. To group the messages by the error level, select the **Group messages** check box. Otherwise, the messages are listed as they are issued.
4. To open the first occurrence of a message, select the **No repeats** check box. If the check box is cleared, each message is opened each time it is issued.
5. To start Designer in Composed Mode when the engine is finished running, select the **View output when complete (if no errors and composing for Exstream viewer or alternative viewer)** check box. You can preview the finished application page by page to see what it looks like after the application is produced.

If you select the **View output when complete (if no errors and composing for Exstream viewer or alternative viewer)** check box, you must take the following into consideration:

- Selecting the **View output when complete (if no errors and composing for Exstream viewer or alternative viewer)** check box works only if there are no errors and the file is being composed for a viewer such as the Exstream Viewer or Output Viewer. Severe errors stop the engine from running. Correct any error messages that cause the engine to stop before proceeding.
- If you are sure the errors have no bearing on the output, clear the check box. If you are not creating output for the viewer, it is also safe to clear the check box.
- The Output Viewer does not automatically open unless an AFP-driven output device is selected in the **Specify output** drop-down list on the **Build Package** dialog box. If no compose file name is given in the **Compose file** box, the output file is automatically named `ExstreamOutput.afp`.

- If you use the Output Viewer, you must also click **Tools > Options** and specify the Output Viewer as the **Alternative output viewer**.

## 2.1.3 Configuring Settings for Message and Report Files



Message and report files are system-generated files that can help you identify and troubleshoot potential problems that can be encountered at engine run time. Customizing the way in which you receive these message files can help you prevent problems when you enter the production stages of application development. Additionally, report files offer you the ability to centralize testing and production data for easy reference.

The **Reports and messages** area lets you specify the type of file and file format you want to use for system-generated files such as message files and report files. The engine creates a file according to the settings you specify and places the file in the specified location. Additionally, you can use the **Reports and Messages** area to name the report and determine remaining content.


You can view any of the message or report files using the following formats:

- A text editor such as Windows Notepad or WordPad.
- An XML editor such as oXygen (message files only).

To configure settings for message and report files, complete the following tasks:

To	Do this
Create a text file that contains the issued messages	<ol style="list-style-type: none"> <li>1. Select the <b>Message file</b> check box.</li> <li>2. To browse to the location of the control file, click  or enter a file name in the box. For example, <code>ExstreamMessages.dat</code>. <code>ExstreamMessages.dat</code> is the default setting.</li> </ol>
Create an XML file that contains the issued messages	<ol style="list-style-type: none"> <li>1. Select the <b>XML Message file</b> check box.</li> <li>2. To browse to the location of the control file, click  or enter a file name in the box. For example, <code>ExstreamMessage.xml</code>.</li> </ol>
Set the message level for the messages	<ol style="list-style-type: none"> <li>1. Select the <b>Message level</b> check box.</li> <li>2. From the drop-down list, select the message level to use: <ul style="list-style-type: none"> <li>• <b>All (info, warning, and error)</b>—View all levels of messages.</li> <li>• <b>Warning and error</b>—View messages that signal problems in the design or engine processing.</li> <li>• <b>Error only</b>—View only messages that signal severe errors.</li> </ul> </li> </ol>



To	Do this
Specify where to create a report file	<ol style="list-style-type: none"> <li>1. Select the <b>Report file</b> check box.</li> <li>2. To browse to the location of the control file, click  or enter a file name in the box. For example, <code>ExstreamReport.dat</code>. <code>ExstreamReport.dat</code> is the default setting.</li> </ol>
Set the report level for the report	<ol style="list-style-type: none"> <li>1. Select the <b>Reporting level</b> check box.</li> <li>2. From the drop-down list, select the report level: <ul style="list-style-type: none"> <li>• <b>None</b>—Does not produce a report</li> <li>• <b>Selection summary</b>—Summarizes information for the entire selection of the application</li> <li>• <b>Customer selections</b>—Summarizes what each customer receives</li> <li>• <b>Customer details</b>—Details what each customer receives and includes record layouts. <b>Customer details</b> produces a large report file. It slows processing and is not recommended for production use. Use this setting when testing only a few customers.</li> </ul> </li> </ol>

The Exstream testing and production engines offer you the ability to specify a text file for engine messages or reports. You can also specify XML as a file type to write engine messages. Generating a message file in XML allows your company to write automated processes using an XML schema provided with the Exstream Design and Production installation. With these automated processes, you can integrate Exstream Design and Production into your existing external business processes, increasing your testing and production workflow.

The XML schema you use to parse an XML message file is available in two locations. You can obtain the schema, `ExstreamMessages.xsd` from your Program Files folder, or from your production environment ZIP folder.

For static and dynamic engine messages, `ExstreamMessages.xsd` uses the following elements and attributes:


#### Elements and attributes for `ExstreamMessages.xsd`

Engine message type	Elements and attributes
Static	<pre>&lt;EngineRunMessage&gt;Static engine message&lt;/EngineRunMessage&gt;</pre> <p>For example:</p> <pre>&lt;EngineRunMessage&gt;Output will be written to Queue files, not the Non-Queue Output file&lt;/EngineRunMessage&gt;</pre>
Dynamic	<pre>&lt;EngineRunMessage&gt;&lt;![CDATA[Dynamic engine message]]&gt;&lt;/EngineRunMessage&gt;</pre> <p>For example:</p> <pre>&lt;EngineRunMessage code="EX002543" level="w" count="1"&gt;     &lt;![CDATA[The package file is down level. The package is at level 800322 and this version of Exstream uses packages at version 800331.]]&gt;&lt;/EngineRunMessage&gt;</pre>

## 2.1.4 Tracking Application Development to Better Manage Campaigns

The **Tracking** area lets you track application development to better manage campaigns. A campaign is a container that holds message objects you send to qualifying customers. With the **Tracking** area, you can enable tracking options for each package file run through the engine. These options function only if you have licensed the Advanced Campaign Management module. To use the options in the **Tracking** area, enable tracking in your campaigns.

To track application development, complete the following tasks:

To	Do this
Store the tracking results used to track campaign distribution	<ol style="list-style-type: none"> <li>1. Select the <b>Track results</b> check box.</li> <li>2. From the drop-down list, select from the following options where you want the results to be stored: <ul style="list-style-type: none"> <li>• <b>None</b>—Results are tracked, but not stored.</li> <li>• <b>File</b>—Stores the results in a text file. During testing, select <b>File</b>, since the engine can produce plain text faster than database information.</li> <li>• <b>Database</b>—Stores the results in a database. Use <b>Database</b> to enable tracking results and response. This database is accessible when you use the <b>Analyze campaign</b> feature.</li> <li>• <b>Delayed database load</b>—Stores the results in a database at the end of an engine run and when the engine run does not produce any errors.</li> </ul> </li> </ol>
Specify where to store the results	<ol style="list-style-type: none"> <li>1. Select the <b>Track file</b> check box.</li> <li>2. To browse to the location of the file, click  or enter a file name in the box.  If you store the results in a file, the default name and location is C:\Program Files\OpenText\Exstream\Exstream &lt;version&gt;\ExstreamTrack.dat.  If you store results in a database, the DSN of that database is updated each time you (or the system administrator) updates the system.</li> </ol>
Control options for <b>Tracking input</b>	<ol style="list-style-type: none"> <li>1. Select the <b>Tracking input</b> check box.</li> <li>2. If you are targeting campaigns based on previous campaigns, you must select from where the input is coming. From the dropdown list, select from the following options: <ul style="list-style-type: none"> <li>• <b>Database</b>—The design engine reads the information from the database and processes the information based on properties specified on the campaign <b>Targeting</b> tab.</li> <li>• <b>None, disable campaigns</b>—The <b>None, disable campaigns</b> option disables the Tracking database and campaigns that use the tracking database.</li> <li>• <b>None, enable campaigns</b>—The <b>None, enable campaigns</b> option disables the Tracking database, but enables campaigns that use the tracking database.</li> <li>• <b>None, error</b>—The design engine does not read the Tracking database and you receive an error message.</li> </ul> </li> </ol>

## 2.2 Setting Options for Testing Tools


After you set the engine options for your application, you can set options for testing tools using the **Run the Engine** dialog box. You set these options in the **Testing** area, which is active only if you have licensed the Rule Analyzer module or the Test Data Capture module and in the **Debug/trace** area. The **Debug/trace** area is available with the base installation of the Exstream design environment.

### 2.2.1 Locating and Identifying Errors in an Application

A basic but powerful testing feature available on the **Run the Engine** dialog box in Design Manager is the **Debug/Trace** area, which offers you the ability to locate and identify errors in an application. The **Debug/Trace** area offers two main tools: a trace capability and a watch on key objects. Debug is a feature that tracks variables and rules in a debug file that you can use to troubleshoot applications. Trace is a process used in troubleshooting timing issues that records the activities of the engine during a test run. Since the information is not compiled, you use the **Debug/Trace** area before you move to the production stages. The information that appears in your file depends on the options you select.

The trace/watch/debug tool is the first tool you should use for troubleshooting potential production engine problems. The tool provides great insight into both timing and variable processing issues. It can serve as a rule/ logic analysis tool, as well as outline the process flow of a page. To use this tool, you must first create a file to locate and identify errors in your application.

To create a file to locate and identify errors in your application:

1. Select the **Debug file** check box to specify the location where you want to write the file, or click  to browse to the location where you want to write the file.

The **Trace logic** and **Watch level** options become available.

2. In the box, enter a name for the file. Specify the entire path the first time you use the option.
3. To record the engine processing during a test run, select the **Trace logic** area. You can view or print the generated trace file using a standard text editor. **Trace logic** is useful when tracking timing issues. It also provides a better understanding of the order of events.
4. Select the object(s) to trace.
  - **All**
  - **None**

5. To record data processing during a test run, select the **Trace data** check box. You can view or print the generated trace file using any standard text editor.

**Note:** Because the **All** and **Trace data** options generate large amounts of output, you should use the **All** or **Trace data** only with limited customer sets. If possible, select only the necessary objects in **Trace logic**. Otherwise, the large amount of detail generated might make it difficult to read the report.

6. To record when a change occurs in a watched variable and place a notification in the trace file, select the **Watch level** check box. From the drop-down list, select one of the following options:

- **Variable**—All variables are watched.
- **Rule**—Only variables in rules are watched.
- **Table**—Only variables in tables are watched.


The option is set in two places: on the variable and on the **Run the Engine** dialog box. To successfully record when a change occurs in a watched variable, make sure the appropriate levels have been set on the valuable in the Property Panel.

For more information about variables and setting properties for variables, see *Using Data to Drive an Application* in the Exstream Design and Production documentation.

## 2.2.2 Generating a Rule Analysis Report to Validate Application Rules

If you have licensed the Rule Analyzer module, you can test the execution of rules in your application. Rules are a set of conditions that let you better control how your application uses objects. Generating a rule analysis report helps you validate the inclusion or exclusion of rules. This also lets your organization ensure successful production runs and gives you greater leverage when you troubleshoot or audit an application. You generate a rule analysis report so you can see which rules and which parts of rules are executed and how many times they are executed.

To generate a rule analysis report:

1. Select the **Rule analysis** check box.
2. In the field, to the right of the **Rule analysis** check box, enter the name of the report file, or click  to browse to the location of your report file. The default name of the file is `RuleAnalysisReport.txt`.
3. If you want a detailed report, select the **Detailed** check box. Design Manager includes in the report each line of every rule and how many times the line was executed. If you clear

the **Detailed** check box, the report shows rule names and how many times the rule was included, excluded, and the number of lines unexecuted.


4. From the **Sort by** drop-down list, select how you want to sort the report. Select from the following options:
  - **Unexecuted Lines**—The report includes lines with the least number of unexecuted lines first on the list. This lets you see the rules that do not have representative data to qualify for the rule.
  - **Times Included**—The report includes the least number of included rules first on the list.
  - **Times Excluded**—The report includes the least number of rules excluded first on the list.
  - **Name**—The report includes rules in a list sorted alphabetically by name.

If you select an option other than **Name**, rules have a secondary sort on name.

## 2.2.3 Capturing Test Data to Collect and identify Rules and Customers in Each Statement

If you have licensed the Test Data Capture module, you can capture test data that identifies rules used for each customer and the customers each statement includes. The Test Data Capture module collects a minimal set of test data that fully exercises all statements of the named and unnamed rules included in an application. Capturing test data is an important testing tool because it lets you further customize reporting levels that best fit your future production goals.


To generate a test data capture report:

1. Select the **Capture test data** check box.
2. Enter the name of the customer driver file, or click  to browse the location of the customer driver file.
3. In the **Customers per rule statement** box, enter the number of customers you want to include per rule statement.
4. From the **Reporting level** drop-down list, select the level of reporting to include. Select from following options:
  - **None**—Selecting **None** does not produce a report.
  - **Summary**—The report includes the rule names and identifies if each was fully tested. This is the default setting.
  - **Detailed**—The report includes each statement of each rule and identifies how many

customers were captured for each statement.

- **Audit**—The report includes each statement of each rule and identifies which customers were captured for each statement.

**Note:** The detailed and audit levels of reporting are available only if you have licensed both the Test Data Capture module and the Rule Analyzer module.

5. In the **Report file** box, specify the location where you want to write the test data capture report, or click  to browse to the location where you want to write the file.
6. From the **Optimization level** drop-down list, select the level of optimization you want the engine to use in finding the smallest possible set of customer. Select from the following options:
  - **Low (Fastest)**—This setting tracks the minimum number of customers possible, uses the least amount of memory, and runs the fastest of all three settings.
  - **Normal**—This is the default setting; customer tracking, memory usage, and speed run at a moderate level.
  - **High (Slowest)**—This setting tracks a large number of customers, uses the most memory, and runs the slowest of all three settings.

Higher optimization levels increase memory usage and run time, but can produce smaller customer driver files with test data.

## 2.3 Executing a Test Engine Run

After you have set the appropriate properties and testing features for the package file you are testing, you can execute an engine test run using the **Run the Engine** dialog box. Executing a test engine run serves as the next to last step in ensuring you have an error-free production run. When you execute the test engine run, you can use the options you set for message and report files to troubleshoot and audit your application to fix errors that can cause errors at production engine run time.

To execute an engine test run:

1. On the **Run the Engine** dialog box, click **Run**.

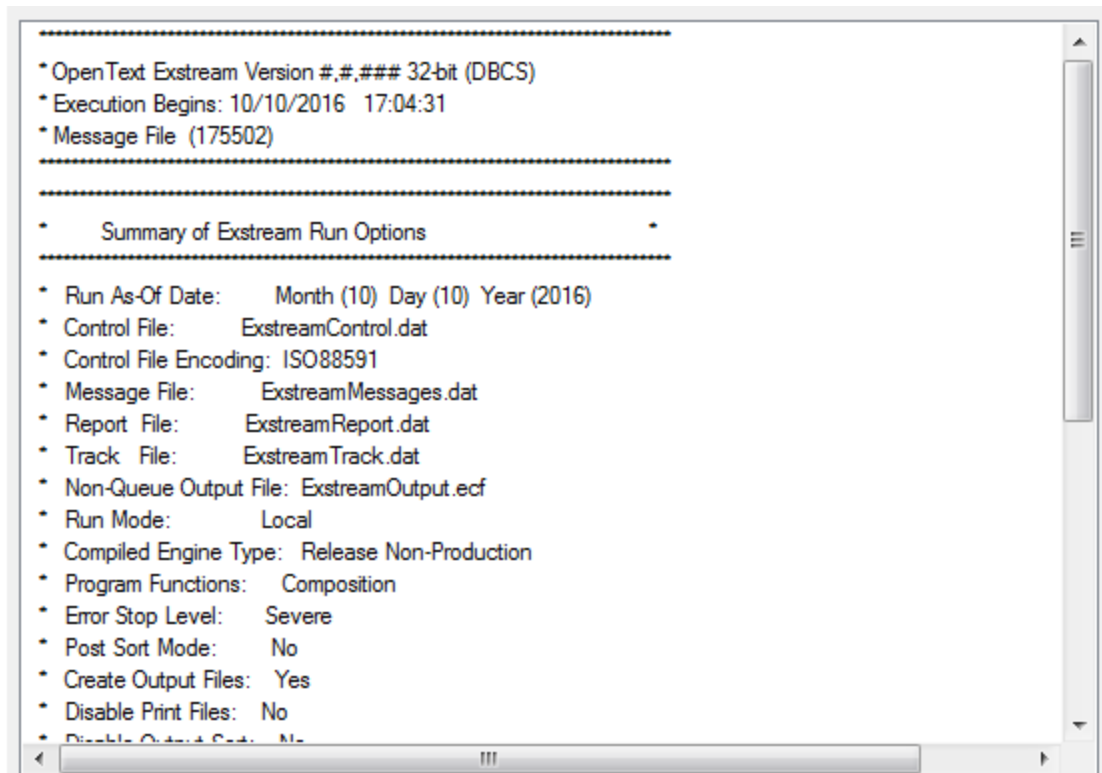
The engine executes a test run. If you select **Ask** from the **When** drop-down list, you receive a message in the **View message file** area.

2. To open the message, click **Yes**.

The **System Report** dialog box opens.

For more information about what you can do with the **When** drop-down list, see [“Customizing How Design Manager Issues Informational Message Files”](#) on page 14.

#### System Report dialog box, Report tab



As you scroll, you see information about the engine run and the informational and error messages. The messages are identified with a number and then a warning level.

3. To see an explanation of a message, click the message. You can view the message on the **Messages** tab.

Messages are identified with a number and a severity level, such as EX001800I. They are set off in a processing section by asterisks. Ignore the leading prefix "EX00" on the message identifiers. Use the four-digit message number to search the Message Dictionary for more information.

The alphabet character refers to the severity level:

Severity level for engine messages

Character	Severity level
I	Informational (or ignore and continue)

Severity level for engine messages, continued

Character	Severity level
W	Warning (or message and continue)
E	Error (or error message and continue)
S	System (or severe message and stop processing)

#### System Report, Messages tab

The screenshot displays the 'System Report, Messages tab' interface. On the left, a tree view under 'Messages' shows a list of error codes: 1000 - NEWCUSTOMER, 1001 - STARTCUSTOMER, 1002 - BEGINCUSTOMER, 1010 - MAXLEVELREACHED, 1011 - SYSTEMEXPIRED, 1012 - SYSTEMEXPIRING, 1013 - NOCOMPOSITIONALLOWED, 1014 - INVALIDCPUID, 1015 - NOTRACKINGALLOWED, 1016 - INVALIDPDL, 1017 - REMINDLANGUAGEFORMU, 1018 - REMINDLOCALEFORMULA, 1019 - ARGLIST\_INVALID, 1020 - MVSINCREMENTAL, 1022 - NOVIERONMVS, 1023 - MULTIPDLGRAPHICS, and 1024 - MULTIPDLRESOLUTION. The right pane, titled 'Message Properties', contains three text input fields: 'Message:', 'Report', and 'Action:'. Below these fields are three labels: 'Severity:', 'Repeats:', and 'Level:'. At the bottom of the interface is a 'Search' section with a text input field and two buttons: 'Find All' and 'Clear'.

4. To continue viewing messages, click the **Report** tab, or click the **OK** to close the **System Report** dialog box.

## 2.4 Validating Application Changes

After you execute a test engine run, the final step in testing your applications is to validate application changes and development using the available Exstream comparison tools. Exstream comparison tools offer you the ability to visually pinpoint errors in your design using



Designer. Although it is important to use the message and report files to perform an in-depth analysis of your applications, the Exstream comparison tools offer you the extra advantage of visual analysis. A visual testing tool can help you identify design issues and allow you to make changes in the design stages. Identifying design problems with Exstream comparison tools greatly improves workflow in future design and application development. Exstream offers two comparison tools to view and compare applications:

- **Exstream Compare tool**—Visually compare Exstream Compose Files (ECF) files in Designer.
- **Exstream Batch Compare tool**—Compare ECF files from the command prompt. This requires you to license the Exstream Batch Compare module.

The Exstream Compare and Exstream Batch Compare utilities compare design objects for structural as well as visual changes. Since drawing routines can vary between versions, the compare utilities might indicate that objects have changed, but they appear visually identical. Depending on the design objects in your application, you might have objects appear as changed due to enhancements and optimizations between versions. The more versions between the two comparisons, the more likely this is to happen.

**Tip:** Before you make changes to the application or upgrade the software, first make a copy of your database.

In addition, when Design Manager composes the files, all portions of a design object are flagged as changed even if a portion of the object has not changed. For example, if you changed one word in a text box, the compare utilities flag the entire text box.

If an application undergoes major changes, the Exstream Compare and Exstream Batch Compare utilities do not resynchronize. For example, if you removed an object, the compare utilities might flag additional objects as well as the deleted object.

To compare ECF files, you must have the following:

- An ECF generated from the original application
- A second ECF generated from the application with changes or in an upgraded version of Exstream
- Access to the database used to generate the ECF files. If you used two databases to compare releases, you must have access to the one with the latest version.

For more information about the Exstream Batch Compare utility, see [“Comparing and Viewing Exstream Batch Compare Files” on page 143](#).

## 2.4.1 Viewing and Comparing Applications in Designer

Exstream Compare lets you view and compare two applications in Designer to track changes and/or errors at the design level. When you execute an engine run, Exstream generates an ECF file for the Exstream Viewer. Before you can perform a comparison, package and run the original application in your usual method. Make your design or version changes, and package and run the application again. To prevent overwriting your first ECF file, give the new file a different name when you run the application the second time.

After generating the ECF files, you can compare the two files:

1. Open Designer with the same database containing the application you used to generate the ECF files.

The comparison uses components from the database. If you used two databases for version comparison, use the newer release.

2. From the **File** menu, select **Compare Compose Files**.

The **Open** dialog box opens.

3. From the **Open** dialog box, highlight the original ECF file.

4. Click **Open**.

The **Open** dialog box remains open.

5. Highlight the second ECF file.

6. Click **Open**.

Designer compares the files and you receive a message showing the number of pages changed, changes made to draw order, and the pages added or removed.

7. To close the message, click **OK**.

The comparison opens in Designer.

8. In Designer, use the Compose File toolbar to browse through your compare files, or click



on the toolbar to open the **Compose File Navigator** dialog box and view the **Composed Customer List**.

You can view the differences overlapping on one page or as a side-by-side view. Both options use a common set of colors to make it easy to identify the differences. The application appears in three different colors:

- **Gray**—Content that did not change
- **Green**—Content in the first ECF file opened that does not match

- **Red**—Content in the second ECF file opened that does not match

Red and green appear only if differences are found in the comparison. The entire file appears gray if no differences are found. Green text does not necessarily correspond to the earlier version of an ECF file, depending on which file you open first.

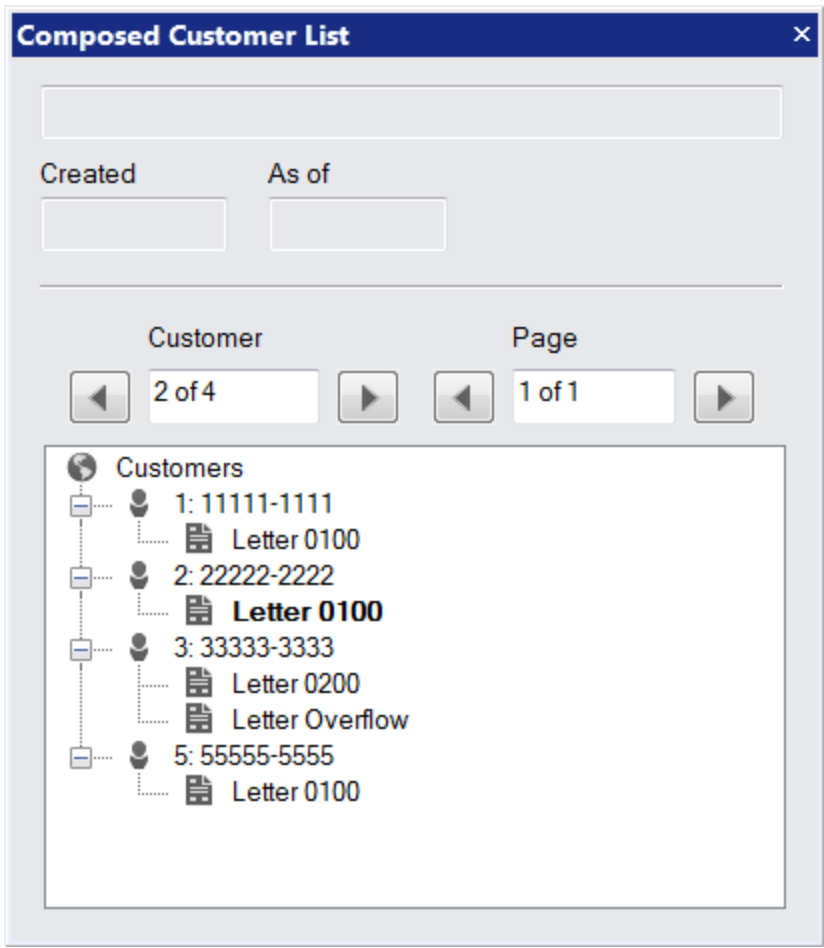
## Choosing the Best Method to Browse Your Exstream Compose Files

After you open your ECF files in Designer, you can choose the best method to view your ECF files. To accommodate the diverse processes of each organization, Designer offers you the ability to browse ECF files using the Compose File toolbar, or from the **Composed Customer List** on the **Compose File Navigator** dialog box. When Designer opens with the compose file in view, the **Compose File Navigator** dialog box and the Compose File toolbar are active.

The **Compose File Navigator** dialog box offers you enhanced navigation abilities. For example, if you are comparing applications with a large amount of customer records and you only want to check changes on one customer, the outline-like view lets you navigate directly to the customer record you must check.

From the **Composed Customer List** on the **Compose File Navigator** dialog box, you can double-click the page to which to navigate. You can also use the **Customer** and **Page** boxes to browse using the arrows or enter the number of the customer or page where you want to browse.

Compose File Navigator dialog box



The Compose File toolbar offers you several buttons that let you browse the entire application page by page or customer by customer.

Compose File toolbar



The pages of the composed files are color-coded to inform you of differences between each object in the two composed files. The following table lists the colors and describes their meaning.

Colors in Compose File Navigator dialog box

Color	Represents
Black	The object is identical in both files.

Colors in Compose File Navigator dialog box, continued

Color	Represents
Blue	The drawing routine for the object has changed. There is no visual difference.
Red	There is a visual difference in object.
Purple	There are a different number of pages in files.
Yellow	There are a different number of customers in files.

## 2.4.2 Viewing AFP or Metacode Compare Files for Testing and Comparison

If you want to view AFP or Metacode files for testing and comparison, you can view the code for AFP or Metacode output in the Edit Panel with a type of data file called File viewer. Using the File viewer data file to view AFP or Metacode output lets you view the output file code for AFP or Metacode so you can validate changes in your application before you send the files to production.

To view AFP or Metacode output:

1. Create a new data file in Design Manager.
2. From the **File type** drop-down list, select **File viewer**.
3. From the **File format** drop-down list, select **Metacode file** if you want to view Metacode output or select **AFP file**, if you want to view AFP output.

For more information about how to create a data file, see *Using Data to Drive an Application* in the Exstream Design and Production documentation.

4. Click **Finish**.

The data file opens in the Property Panel for you to define.

5. On the **Advanced** tab, from the **Record Type** drop-down list, select the blocking method used to create the file (The viewer fails with the default setting of **None**).
6. On the **Test Data Source** and **Production Data Source** tabs, specify the path name to the external output file in the bottom box.
7. Save the data file.
8. To view the file contents, drag the data file to the Edit Panel.
9. To select a different view of the output contents, right-click anywhere in the Edit Panel and select a view. Select from the following options:

- **Hex dump**
- **Explanation**
- **Raw Data**
- **Offsets** (None, Hexadecimal, Decimal)

## 2.5 Viewing Package Reports

A package report can help you validate the information contained within the application(s) in your package file. Additionally, a package report notifies you of any errors that occur during the packaging process, which can help you troubleshoot potential issues before you move to the production stages. The package report feature provides one location to view the objects in and details about an existing package file.

To view the package report:

1. In Design Manager, select **Tools > Report package contents**.

The **Select a Package or Resource File** dialog box opens.

2. Select the package file and click **Open**.

The **Package Report** dialog box opens.

3. Click **OK**.

If there are any package file errors, you receive an error message.

4. Click **OK**.

The **Package Report** dialog box opens. The following table references the information contained in the **Package Report** dialog box.

Area	Package file information
<b>Package</b>	<p>The following information appears in the <b>Package</b> area:</p> <ul style="list-style-type: none"><li>• Package file location</li><li>• <b>Date created</b>—Date and time the package file was created</li><li>• <b>Created by</b>—User who created the package file</li><li>• <b>Revision #</b>—Number of revisions to the package file and the package version number</li><li>• <b>Last revised</b>—Date and time the package file was last revised</li><li>• <b>Device</b>—Output driver packaged for the application</li><li>• <b>OpenText Exstream version</b>—Release version at the time of packaging</li></ul>

Area	Package file information
Application	<p>The following information appears in the <b>Application</b> area:</p> <ul style="list-style-type: none"><li>• <b>Name</b></li><li>• <b>Revised</b></li><li>• <b>Version</b></li></ul>
Object Version	<ul style="list-style-type: none"><li>• The following information appears in the <b>Object Versions</b> area:</li><li>• <b>Date</b>—Effective date used to qualify objects for packaging</li><li>• <b>Version</b>—Earliest object version included in packaging</li><li>• <b>Jurisdiction</b>—Jurisdiction included in package file:<ul style="list-style-type: none"><li>• <b>Ignore</b> if no jurisdictions were included.</li><li>• <b>Specified</b> if a single jurisdiction was included.</li><li>• <b>All</b> if all jurisdictions were included.</li></ul></li></ul>
Object Details	<p>The following information about each object in the package file appears in the <b>Package Report</b> dialog box:</p> <ul style="list-style-type: none"><li>• Object type</li><li>• Object name</li><li>• User who last updated the object</li><li>• Object version</li><li>• Date the object was last updated</li><li>• Time the object was last updated</li></ul>

5. To exit the dialog box, click **OK**.

# Chapter 3: Packaging an Application

Packaging an application is the first step to producing output. Creating a package file for your application(s) lets you combine all of the objects or elements into one format that is optimized for processing.

A package file is the connection between the design environment and the engine and contains the following information that the engine requires to process an entire application:

- Data file mapping
- All design objects in your application (for example, documents, campaigns, pages, variables, and rules)
- All required resources (for example, fonts and images)
- Pre-composed versions of the static design objects and resources

The package file contains only the information about the objects that is necessary for the engine to run. As a result, you cannot recover an application from a package file.

This chapter discusses the following topics:

- [“About Packaging an Application” below](#)
- [“Packaging Applications in Design Manager” on page 34](#)
- [“Packaging Applications From the Command Prompt” on page 53](#)
- [“Increasing Productivity with a Reusable Packaging Format” on page 71](#)
- [“Integrating Changes at the Application Level” on page 72](#)
- [“Optimizing the Packaging Process Through Consolidating Applications” on page 75](#)
- [“Implementing Collaborative Authoring Within the Packaging Process” on page 75](#)
- [“Managing Application Objects With Version Labels” on page 76](#)
- [“Packaging switches” on page 59](#)

## 3.1 About Packaging an Application

One package file is created for each application. You can package an entire application (creating a complete package file) or package incrementally (only those components that have changed since the last package date). During packaging, the software pre-composes many of the design elements and resources in an application. Packaging reduces the processing time during production. You can also create package profiles. By saving your packaging settings, you can



reuse the settings instead of having to re-configure them each time you create a package file. Package profiles also let you configure packaging settings for other users. For more information about package profiles, see [“Managing Package Profiles” on page 48](#).

Incremental packaging is available only if the system configuration includes the **Incremental packaging** option on the **Workflow** tab of the **System Configuration** properties in **Environment > System > System Settings > Basic tab**. For more information about enabling features and functions in the **System Configuration** properties, see *System Administration* in the Exstream Design and Production documentation.

**Note:** If your application requires pre-packaging, incremental packaging is not supported for DBCS applications.

Additionally, you can choose to package specific non-design and environmental objects into a sub-package file. At run time, the objects in the sub-package file override corresponding objects in the main package file, allowing you to make changes or updates to objects without repackaging an entire application, which can be costly and time-consuming during production.

For example, suppose you distribute a telecommunications application in an effort to achieve several objectives at once. Among these objectives, your sales team wants to include a single campaign to sign new customers by way of in-store registration. Each customer is to receive a specific barcode that, when scanned, automatically creates an account. After the design team creates the main package file, the sales team requires that the barcode be changed. Without repackaging the application, the design team creates a new single campaign with only the new barcode object, creates a sub-package file, and initiates an engine run. During processing, the engine appends the contents of the sub-package file to the main application and the updated barcode replaces the existing barcode.

For more information about sub-package files, see [“Creating Sub-Packages for Campaigns, Documents, and Objects” on page 73](#).

Exstream offers two methods to package single or multiple applications. You can complete the packaging process using one of the following methods:

- **Using Design Manager**—The Design Manager user interface lets you perform all of the tasks necessary to package an application from the design environment. This method is helpful to designers and remote users who must ensure that the application contains all the necessary design elements, rules, and functions to produce the desired output.

For more information about using Design Manager to package an application, see [“Packaging Applications in Design Manager” on the next page](#).

- **Using the Windows command prompt**—Using the Windows command prompt is ideal for developers or system administrators who want to package applications outside of the Exstream design environment. For example, if you only need to run an updated package file, you can do so strictly from the command prompt.

For more information about using the Windows command prompt to package an application, see [“Packaging Applications From the Command Prompt” on page 53](#).

To streamline the packaging process and improve productivity, you can perform additional tasks (for example, [uploading your packages to a common repository](#), [creating package profiles](#), or [using version label management](#)) during the packaging process.

## 3.2 Packaging Applications in Design Manager

Packaging applications in Design Manager is the preferred method for designers and/or remote users who must test and preview output for an application. Using the Design Manager interface lets a user validate the inclusion of the design elements, rules, and functions that are required in order to produce the desired output. Before you begin packaging tasks, make sure that your application includes the appropriate resources and data mappings in order to avoid potential packaging errors.

This section discusses the following topics:

- [“Creating and Customizing a Package File” below](#)
- [“Uploading a package file to the CAS repository” on page 41](#)
- [“Managing Object Versions in Your Package File” on page 44](#)
- [“Specifying Resource Packaging Options” on page 46](#)
- [“Managing Package Profiles” on page 48](#)
- [“Packaging for Previews ” on page 49](#)

### 3.2.1 Creating and Customizing a Package File

In Design Manager, you set the packaging options on the **Build Package** dialog box for the application that you want to produce. When you set the packaging options for an application, you are preparing that application for publishing, which includes packaging, creating package profiles, and previewing the finished publication.

### Build Package dialog box

The screenshot shows the 'Build Package' dialog box with the following settings:

- Application:** Explanation of Benefits
- Profile:** No Profile
- Target:** Application
- Type:** Complete package
- Package file:** C:\Applications\Out\EOB.pub
- Output options:**
  - ☐ Create for DBCS output queue device(s)
  - ☒ Create for SBCS output queue device(s)
  - ☐ Specify output (SBCS) Exstream Viewer
- Version method:** Version status
- Includes selected status and any statuses above it:**
  - ☐ Approved
  - ☒ Submitted for Approval
  - ☐ Work in Progress
  - ☐ Rejected
- Substitute Quick Fix versions:** ☐
- Include versions from the following user:** admin
- Effective date:**
  - From:** 2018-01-01
  - To:** 2019-12-31
  - Date range:** 11:13
- Jurisdictions to include:** Ignore
- No Jurisdiction**
- ☐ Report objects excluded from jurisdictions when packaging

Buttons: OK, Cancel

To create a package file in Design Manager, complete the following steps:

1. In Design Manager, click **Tools > Package....**

Alternatively, you can right-click an application in the Library and select **Package....**

The **Build Package** dialog box opens.

**Note:** When packaging a DBCS application, the pre-packaging utility automatically runs. The pre-packaging utility uses your test data sources, active Windows code page, and formula calculations in order to determine the characters that are included in the package file. Make sure that your test data sources, or objects that are used in the pre-packaging engine run for your application, include all of the DBCS font characters that are required in subsequent engine runs. Also, if the paths for output are not specified correctly, or if there are characters missing from the package file, you receive an error message.

For information about using font resource management to specify the DBCS font characters that are included in the package file, see *Importing External Content* in the Exstream Design and Production documentation.

2. In the **Application** box, click .

The **Select Application** dialog box opens. To select an application, do the following:

- In the **Look in** box, select the folder that contains the application that you want to package and click OK.
- In the **Name** box, type the name of the application you want to package, or select the application from the selection list and click OK.

Keep in mind that if you selected your application from the Library, the application name is shown in the **Application** box and you do not need to select an application again on the **Select Application** dialog box.

3. You can package your application without a profile, you can select a previously saved package profile from the **Profile** list (if a profile is available), or you can select **<Create new profile>** from the list. By default, the **No Profile** option is selected.

If you select **<Create new profile>**, the **Create New Package Profile** dialog box opens when you click **OK** on the **Build Package** dialog box. To create a new package profile and continue packaging your application, enter the name that you want to use for the new profile, and click **OK**.

For more information about package profiles, see [“Managing Package Profiles” on page 48](#).

4. From the **Target** list, select the objects (targets) you want to include in your package file:

Target	What is included	When to use
Application	Complete application, including all documents and campaigns	Use the first time an application is packaged (default)
Campaigns	Campaigns within a previously packaged application	Use when changes have been made to multiple campaigns, but the rest of the application is unchanged

Target	What is included	When to use
Documents	Documents within a previously packaged application	Use when changes have been made to multiple documents, but the rest of the application is unchanged
Non-Design Objects	Non-design and environmental objects within a sub-package file	Use when changes have been made to non-design or environmental object (s), but the rest of the application is unchanged
Single Campaign	Specific campaign within a previously packaged application	Use when changes have been made to a single campaign, but the rest of the application is unchanged
Single Document	Specific document within a previously packaged application	Use when changes have been made to a single document, but the rest of the application is unchanged

5. From the **Type** list, select the type of package file to create:

Type	What is included	When to use
Complete package	All design elements	Use when packaging for most output drivers (default)
Complete package without resources	All design elements except fonts and images	Use if you are using a separate resource library file, or if your output device fonts are the only fonts used, or if you are using inserts only. This option saves processing time and reduces package file size.  (You must select either AFP or PostScript as the printer definition language (PDL). If the AFP or PostScript options are not selected, Exstream Design and Production issues an error message).
Build print resource files	Application resource information	Use if font specifications or other resources have changed on your system.  If you specify this option on the <b>Type</b> list, the <b>Resource Packaging</b> dialog box opens after you click <b>OK</b> on the <b>Build Package</b> dialog box. You must specify your resource file for output in the <b>Resource Packaging</b> dialog box.  For more information about specifying print resource files, see <a href="#">"Specifying Resource Packaging Options" on page 46</a> .  (You must select either AFP or PostScript as the printer definition language (PDL). If the AFP or PostScript options are not selected, Exstream Design and Production issues an error message).

6. In the **Package File** box, enter the path and name of the package file you are creating. By default, your database path and application name are used as the name of the file. For example, if your application is called **Test Application**, the location of your database is **C:\Test Databases**. The file is **C:\Test Databases\Test Application.pub**.

If you enter only a file name, the default file location is as follows in Windows 7:

**C:\Users\<user name>\Documents\OpenText\Exstream\Exstream <version>**

7. To specify how you want to create output when you execute a production engine run, select one of the following options:

To	Do this
Create a package that uses the output queue defined in the application	<p>Select <b>Create for output queue device(s)</b>.</p> <p>This option lets you uniquely determine how output is created and processed.</p> <p>The <b>Create for output queue device(s)</b> option appears only if you have selected either <b>SBCS</b> or <b>DBCS</b> from the <b>Application mode</b> list on the <b>Workflow</b> tab in <b>Environment &gt; System &gt; System Settings</b>.</p> <p>You must license the High-Volume Delivery module to use this feature.</p>
Create a package that uses an SBCS output queue device	<p>Select <b>Create for SBCS output queue device(s)</b>.</p> <p>If you have selected <b>SBCS/DBCS</b> from the <b>Application mode</b> list on the <b>Workflow</b> tab in <b>Environment &gt; System &gt; System Settings</b> and you want to create output on an SBCS output queue device, you must select <b>Create for SBCS output queue device(s)</b>.</p> <p>This option is not available if you have selected either <b>SBCS</b> or <b>DBCS</b> from the <b>Application mode</b> list.</p>
Create a package that uses a DBCS output queue device	<p>Select <b>Create for DBCS output queue device(s)</b>.</p> <p>If you have selected <b>SBCS/DBCS</b> from the <b>Application mode</b> list on the <b>Workflow</b> tab in <b>Environment &gt; System &gt; System Settings</b>, and you want to create output on a DBCS output queue device, you must select <b>Create for DBCS output queue device(s)</b>.</p> <p>This option is not available if you have selected either <b>SBCS</b> or <b>DBCS</b> from the <b>Application mode</b> list.</p>
Create a package that uses a specific user-selected output device	<p>Select <b>Specify output</b>.</p> <p>You must also select an output device from the list of available output devices.</p> <p>To select an output device:</p> <ol style="list-style-type: none"> <li>Click in the box to the right of the <b>Specify output</b> option.</li> </ol> <p>The <b>Select Output</b> dialog box opens.</p> <ol style="list-style-type: none"> <li>Select the output device from the list of available output devices you want to use and click <b>OK</b>.</li> <li>Select the resolution for your output device from the <b>Resolution</b> list.</li> </ol> <p>Keep in mind that output creation and processing is determined by the output device so all applications that use this output device are processed in the same way.</p>

8. Select the **Close summary panel if no errors** check box to close the packaging summary panel if no errors occur while packaging your application.

If you also select the **Run Engine when complete** check box, then the summary panel is closed if no errors occur during both packaging and the engine run.

9. If you have previously packaged the selected application, select the **Update existing package** box to update the existing package file with only the objects that have changed.


Updating the existing package file is useful when you are working with large package files since you do not have to include all files in the packaging process.

**Important:** Do not change the name of the package file. If you change the name, the file will not be updated.

You must also ensure that the output driver properties have not changed if you choose to update the existing package file. If they have, you must repackage the application.

If you are using a package profile, and you select the **Update existing package** check box, then the **Profile** list reverts to the **No profile** option and the package settings are no longer saved as a package profile. If you want to save the updated settings as a package profile, then you must create a new package profile by selecting **<Create new profile>** from the **Profile** list.

10. If you want to run the engine after packaging is complete, select the **Run Engine when complete** check box.

To modify the settings for your engine run, click  to open the **Run the Engine** dialog box, and update the settings as required. By default, the engine run uses the last settings that were entered on the **Run the Engine** dialog box.

If you want to package your application without running the engine, clear the **Run Engine when complete** check box. Clearing the **Run Engine when complete** check box might make it easier to find any error messages because you can troubleshoot the package file first. You can then run the engine manually in a second step by selecting **Tools > Run Engine** from the Menu bar.

For more information about the **Run the Engine** dialog box, see [“Testing Applications Using the Design Engine” on page 10](#).

11. Select the **Register package date** check box to register the creation date and time for your production package file in the application database.

Each time you package the application with this check box selected, the time stamp is overwritten with the new creation date and time. You can retrieve the date and time by viewing the application in the Property Panel.

12. If you want to include messages about updates to images that are inserted from a connected common asset service (CAS) repository, select the **Check CAS for image updates** check box.

For more information about connecting a CAS repository, see *System Administration* in the Exstream Design and Production documentation.

13. If you want to upload the package file you are creating to a connected CAS repository, select the **Upload to CAS when complete** check box.

For detailed information about configuring settings for the package file you are uploading, see [“Uploading a package file to the CAS repository” on page 41](#).


14. If you want to compress the package file, select the **Compress package contents** check box. Compression takes place at the end of the packaging process, and compresses the contents that are located between the package header and the AQS manifest. This setting is recommended for package files that will be uploaded to OpenText cloud applications.

Keep in mind the following considerations when using package file compression:

- Package file compression is enabled only when the **Build Package Target** is set to **Application**, and the **Type** is set to **Complete package** (incremental packaging is not supported).
- Compressed package files can be decompressed by the Windows (32-bit and 64-bit), Linux (32-bit and 64-bit), or AIX (64-bit) versions of the Exstream test engine or production engine.
- Although you should always compress Design and Production package files that are used with OpenText cloud applications, package file compression can also be used to reduce the amount of resources required to store package files locally or transmit them to other platform resources such as CAS.
- Package files that contain mostly images might not achieve a high rate of compression, because images are already in a compressed format.

**Tip:** The Exstream engine decompresses compressed package files at run time. The default maximum size for a decompressed package file is 104857600 bytes (approximately 100 MB). You can change the default maximum size using the MAX\_DECOMPRESSED\_PACKAGE\_BYTES engine switch. Because the engine run will halt if there is not a temporary directory available to store a decompressed package that exceeds the maximum size, it is a good idea to also use the TEMPORARY\_DIRECTORY engine switch any time you use package compression.

For more information about using these switches, see *Switch Reference* in the Exstream Design and Production documentation..

15. To specify which version(s) of objects to include in your package files, click  and update the settings in the expanded **Build Package** dialog box.

For more information about object versions, see [“Managing Object Versions in Your Package File” on page 44](#).

16. Click **OK** to create your package file.

The **Build Package** dialog box closes and the **Build Production Package File** dialog box opens.

17. The progress bar on the **Build Production Package File** dialog box indicates the status of your package file.

If you selected the **Run Engine when complete** check box, an engine run is initiated after packaging is complete.



If you selected the **Upload to CAS when complete** check box, the created package file is uploaded to the CAS repository.

18. Click **Close**.

The **Build Production Package File** dialog box closes. You can access the created package file from the location you specified in the **Package file** box.

## 3.2.2 Uploading a package file to the CAS repository

The common asset service (CAS) provides a centralized versioned repository for storing package files and allows a user to upload multiple versions of a package file along with associated metadata. You can upload your package files to the CAS from Design Manager or from the command line.

Uploading your package files to the CAS also allows you to select and use the stored package files in the integrated Exstream platform environment and take advantage of the production engine and output delivery orchestration features. You can use the stored package files in Communications Builder and run the Exstream production engine from Control Center.

For complete information about using Exstream package files in Communications Builder, see *Creating Exstream Engine plug-ins* in *OpenText Exstream: Communications Builder Configuration Guide*. For complete information about setting up a Communications Server application and running the Exstream production engine from Control Center, see *Setting up Communications Server applications* in *OpenText Exstream: Communications Server Administration Guide*.

Before you can upload package files to the CAS, you must configure your management gateway connection settings on the **Integration** tab in **Environment > System > System Settings** in Design Manager. For information about connecting to the management gateway, see *System Administration* in the Exstream Design and Production documentation.

**Note:** If you are using a multi-tenant system, make sure that you use the correct **OTDS base URL**, **Resource ID**, **Tenant name**, and **Application domain** settings for the tenant to which you want to connect.

To build a package file and upload it to the connected CAS repository, complete the following steps:

1. In Design Manager, click **Tools > Package...**

Alternatively, you can right-click an application in the Library and select **Package...**

The **Build Package** dialog box opens.

2. In the **Application** box, click  .

The **Select Application** dialog box opens. To select an application, do the following:

- a. In the **Look in** box, select the folder that contains the application that you want to package and click **OK**.
- b. In the **Name** box, type the name of the application you want to package, or select the application from the selection list and click **OK**.

Keep in mind that if you selected your application in the Library, the application name is shown in the **Application** box and you do not need to select an application again on the **Select Application** dialog box.

3. Set the packaging options on the **Build Package** dialog box for this application.

For more information about customizing packaging options, see [“Creating and Customizing a Package File” on page 34](#).


4. Select the **Upload to CAS when complete** check box to upload the package file to the CAS after packaging is complete.

Note that if you select **Build print resource files** from the **Type** list, or if you are packaging non-design objects, the **Upload to CAS when complete** check box is disabled.

If you previously entered a path for the archived package file in the **Archived package file directory** box in **Tools > Options...**, a copy of your package file will also be saved to the package archive directory.

Also keep in mind that selecting the **Upload to CAS when complete** check box also selects and disables the **Register package date** check box. Selecting the **Register package date** check box saves the creation date and time for your package file in the application database.

**Important:** If you have not entered the management gateway connection settings on the **Integration** tab in **Environment > System > System Settings**, or if you are not connected to the management gateway as an authorized OTDS user, the **Upload to CAS when complete** check box is not visible on the **Build Package** dialog box. You might need to contact your system administrator for assistance in configuring the management gateway connection settings.

5. Click  next to the **Upload to CAS when complete** check box to configure the options for the package file to be uploaded.

The **CAS Package Details** dialog box opens.

6. In the **Name** box, enter the name for your package file as you want it to appear in the CAS.

By default, this box is populated with a default name generated for your application based on the packaging options that you select on the **Build Package** dialog box.

For example, if you select **Single Campaign** from the **Type** list, the name of the object being packaged is appended as a prefix to the name of the package file uploaded to the CAS (that is, `SingleCampaignName - ApplicationName`).

**Tip:** You can upload multiple package files with the same name to the CAS. To avoid confusion, it is recommended that you use a unique name when uploading a new package file to the CAS .

7. In the **Description** box, enter an optional description for your package file.

By default, this box is populated with the description provided in the application properties in Design Manager.

8. From the **Error stop level** list, select the packaging error level at which you want Design Manager to cancel the CAS upload.
9. To upload your package file to the CAS, select **Create a new package**.

This option creates a new package file in the CAS. If this is the first time this package file is being uploaded to the CAS, then this option is selected by default.

10. To update an existing package file in the CAS with the package file you are uploading, select **Upload a new version of an existing package**.

If the application you are packaging has been previously uploaded to the repository then this option is selected by default, and the corresponding package file is selected in the list of existing package files available in the CAS.

Keep in mind that when you update an existing package file, a new version of the package file is created in the CAS, and the version number of the package file stored in the repository is also incremented to indicate that it was updated. The previous versions of the updated package files remain available in the CAS.

11. When you select the package file you want to update, the following information about the selected package file is displayed in the **Package Information** area:
  - **Design Manager Version**—The version of Design Manager that was used to package the application.
  - **Application**—The name of the application that was packaged.
  - **Package Target**—The objects (targets) included in the package file that is uploaded. This attribute corresponds to the option selected from the **Target** list on the **Build Package** dialog box.
  - **Package Type**—The type of package file that is uploaded. This attribute corresponds to the option selected from the **Type** list on the **Build Package** dialog box.
  - **Package Version Status**—The version status of the package file that was uploaded, which indicates the version(s) of objects included in the uploaded package file. This

attribute corresponds to the option selected from the **Versions** area on the expanded **Build Package** dialog box.

12. Click **OK** to apply your settings.

You are returned to the **Build Package** dialog box where you can continue to customize your package file.

13. Click **OK**.

The **Build Package** dialog box closes and the **Build Production Package File** dialog box opens.

14. The progress bar on the **Build Production Package File** dialog box indicates the status of your package file. When your package file is successfully uploaded to the CAS, you receive a packaging message notifying you that the upload is complete.


**Tip:** By default, Design Manager waits for 10 minutes for a package file to finish uploading to the CAS before the upload times out. You can specify a custom network timeout to change the amount of time that Design Manager waits for a package file to finish uploading before canceling the upload.

To specify the custom timeout, in Design Manager, from the Menu bar, click **Tools > Options...** and in the **CAS upload timeout (seconds)** box, enter the amount of time (in seconds) to wait for a package file to finish uploading to the CAS.


15. Click **Close**.

The **Build Production Package File** dialog box closes.

### 3.2.3 Managing Object Versions in Your Package File

When an object undergoes changes, different versions of the object are available for packaging without repackaging the application. Use the **Versions** area to specify which version(s) of objects to include your package files. To access the **Versions** area of the **Build Package** dialog box, click .

To manage object versions in your package file, complete the following tasks:

To	Do this
Specify how the packaging process will select a version or versions of an object to place in the package file	<p>From the <b>Version method</b> drop-down list, select one of the following options:</p> <ul style="list-style-type: none"> <li>• <b>Version status</b>—Select object versions based on their version status. When you select this option, the <b>Includes selected status and any statuses above it</b> slider appears.</li> <li>• <b>Custom state</b>—Package objects with a particular custom approval state and objects with a status of Approved. You must license the Advanced Design Workflow module to use this feature. When you select <b>Custom state</b>, the <b>Approval state</b> drop-down list appears below the <b>Version method</b> drop-down list. From the <b>Approval state</b> drop-down list, select a custom approval state.</li> <li>• <b>Version Label</b>—Package objects for a particular version label. This option is available only if you create a version label for the application. When you select <b>Version Label</b>, the <b>Version label</b> drop-down list appears below the <b>Version method</b> drop-down list. From the <b>Version label</b> drop-down list, select the label.</li> </ul> <p>For information about creating version labels for applications, see <a href="#">“Selecting a Version Label When Packaging an Application” on page 80</a>.</p>
Limit packaging based on objects in the application with a particular version status	<p>If you have selected <b>Version status</b> from the <b>Version method</b> drop-down list, move the <b>Includes selected status and any statuses above it</b> slider to one of the following options:</p> <ul style="list-style-type: none"> <li>• <b>Approved</b>—Includes all objects with a status of Approved or Archived</li> <li>• <b>Submitted for Approval</b>—Includes all objects with a status of Submitted for Approval, Approved, and Archived</li> <li>• <b>Work in Progress</b>—Includes all objects with a status of Work in Progress, Submitted for Approval, Approved, and Archived</li> <li>• <b>Rejected</b>—Includes all objects with a status of Rejected, Work in Progress, Submitted for Approval, Approved, and Archived</li> </ul>
Package objects with a Quick Fix status in place of objects with Approved status	<p>Select the <b>Substitute Quick Fix versions</b> check box. The <b>Substitute Quick Fix versions</b> check box is active only if you select <b>Approved</b> on the <b>Includes selected status and any statuses above it</b> slider.</p> <p>For information about Quick Fixes, see <i>System Administration</i> in the Exstream Design and Production documentation.</p>
Package versions from a particular user	<ol style="list-style-type: none"> <li>1. If you have selected <b>Submitted for Approval</b>, <b>Work in Progress</b>, or <b>Rejected</b> on the <b>Includes selected status and any statuses above it</b> slider, select the <b>Include versions from the following user</b> check box.</li> <li>2. Under the <b>Include versions from the following user</b> check box, click  . The <b>Select Design User</b> dialog box opens.</li> <li>3. From the list, select a design user.</li> <li>4. Click <b>OK</b>. The design user that you selected appears in the box under the <b>Include versions from the following user</b> check box. The user is only pertinent for Rejected, Work in Progress, and Submitted for Approval version statuses. Approved and Archived versions no longer have a user associated with them.</li> </ol>

To	Do this
Select object versions as they existed on the specified date or select the most recent version in the range of dates	<p>Use the <b>Effective Date</b> drop-down list to select from the following options:</p> <ul style="list-style-type: none"> <li>• <b>As of Now</b>—Selects the latest valid version of the objects as of the current date (and time, if used).</li> <li>• <b>As of Date</b>—Selects the latest valid version of the objects as of a specified date (and time, if used). You must select the correct date (and time, if used) from the adjacent <b>To</b> box.</li> <li>• <b>Date Range</b>—Selects the valid version of objects as of a range of dates (and time, if used). You must select the correct date (and time, if used) from the adjacent <b>From</b> and <b>To</b> boxes.</li> </ul>
Include jurisdictions on object versions in your package file	<p>If you have licensed the Compliance Support module, you can include jurisdictions on object versions.</p> <p>To include jurisdictions on an object version, you must first select <b>Jurisdictions</b> on the <b>Design Manager</b> tab of the <b>System Configuration</b> settings.</p> <p>After you enable the appropriate settings for jurisdictions, you can select the <b>Jurisdictions to include</b> option on the <b>Build package</b> dialog box and select from the following options:</p> <ul style="list-style-type: none"> <li>• <b>Ignore</b>—Ignore jurisdictions and use latest valid version of objects. If the application does not contain jurisdictions, make sure you select the <b>Ignore</b> option. Otherwise, the engine lists jurisdictions as missing in the package report.</li> <li>• <b>Single Jurisdiction</b>—Include one specified jurisdiction for objects using jurisdictions.</li> <li>• <b>All Jurisdictions</b>—Include all valid jurisdiction versions for all objects.</li> </ul> <p>For information about the Compliance Support module, see <i>System Administration</i> in the Exstream Design and Production documentation.</p>

## 3.2.4 Specifying Resource Packaging Options

If you have selected the **Build print resource files** option from the **Type** list on the **Build Package** dialog box and click **OK**, the **Resource Packaging** dialog box opens.

The **Resource Packaging** dialog box lets you specify which image and font resources are packaged in your package file.

**Note:** This dialog box is available only if the **Standalone resource packaging** option is enabled on the **Workflow** tab of the **System Configuration** properties in **Environment > System > System Settings > Basic tab**. For more information about enabling features and functions in the **System Configuration** properties, see *System Administration* in the Exstream Design and Production documentation.

To specify your resource file for output on the **Resource Packaging** dialog box, you must perform the following steps:

1. From the **Resource creation method** list, select one of the following options to specify how you want to populate the file:
  - **All resources in one file**—Include all the fonts and images in one file. **All resources in one file** is the default selection.

- **One resource per file**—Create a separate file for each image or font.

**Note:** When packaging one resource per file in a DBCS application for AFP output, Design Manager creates 256 font files for each design font and creates all of the resources in the application before pre-packaging.

- **One resource per file if not exist**—Create a file for each image or font if it does not exist from a previous packaging operation.
2. If you select **All resources in one file**, in the **Resource file or directory** box, enter a file name. If **All resources in one file** is not selected, enter the directory where the resources are created.
  3. To exclude image or font resources from the package file, complete the appropriate tasks in the following table:

To	Do this
Exclude image resource files in your application	In the <b>Resources to build</b> area, clear the <b>Build images</b> check box The <b>Build images</b> check box is selected by default.
Exclude font resource files in your application	In the <b>Resources to build</b> area, clear the <b>Build fonts</b> check box The <b>Build fonts</b> check box is selected by default. If you have selected the <b>Build fonts</b> check box, the <b>Build fonts</b> drop-down list becomes available.

4. If you have selected to include font resource files in your application, from the **Build fonts** list, select one of the following options:
  - **All fonts**—Build all the fonts as resource files.
  - **All fonts with names in the name table**—Build only the fonts with names in the name table.
  - **All fonts without names in the name table**—Build only the fonts not listed in the name table.

For more information on the name table of the font properties, see *Creating Output* in the Exstream Design and Production documentation.

**Note:** For applications that contain DBCS characters, Design Manager references the appropriate encoding at the top of a single font resource file or as a separate code page (.cp) file for separate font resource files.

5. To create a resource packaging report detailing the resources included in your application, select the **Resource report** check box.

The **Resource report** field becomes active. By default, the **Resource report** field is populated with the file path for the `ExstreamResource.rpt` file.

**Note:** When creating a resource packaging report for a DBCS application on AFP output, the report includes the 256 AFP fonts created for each design font.

## 3.2.5 Managing Package Profiles

To create a package file, you can manually configure the packaging settings, or you can save the packaging settings in a package profile. You or other users can use the pre-configured packaging profiles to reduce the time it takes to select packaging options and start the packaging process much faster. For example, suppose you create a new package profile with the name `Work In Progress`, which includes the original application with certain objects requiring approval. Next, the application is reviewed, approved, and sent to the production stage. You create a second package profile with the name `Approved`, which only includes previous versions of the approved application. You want to make revisions to the approved application, so you create a second version of the application that you can use to make revisions and resubmit for approval. Another user wants to view only his changes to the application, so the user selects the `Work In Progress` package profile, which includes the original version of the application. Since package profiles can be used with any version of an application, the user adjusts packaging settings for the user name and creates a third package profile named `WIP for J. Smith`, which includes only the user's changes to the application.

Keep in mind the following considerations when managing package profiles:

- You can use package profiles for any version of an application.
- If you delete a package profile, the profile is deleted from all versions of an application.
- You cannot load or unload a package profile in an XOB.
- Design Manager uses the following naming convention for package profiles:

`<Target>, <Version>, <Device>`





For example:

`Campaign, WIP for all, DBCS Queue`

To manage an application's package profile, complete the following steps as needed:

1. In Design Manager, from the Library, drag the application for which you want to manage a package profile to the Property Panel.
2. Click the **Packaging** tab.
3. On the **Packaging** tab, complete the appropriate task in the following table:



To	Do this
Create a package profile	<ol style="list-style-type: none"> <li>1. Click  . The <b>Create Package Profile</b> dialog box opens.</li> <li>2. Change any packaging options as needed for your application and click <b>OK</b>. If you have selected <b>Build print resource files</b> from the <b>Type</b> drop-down list, the <b>Resource Packaging</b> dialog box opens.</li> <li>3. Set the appropriate options on the <b>Resource Packaging</b> dialog box, as needed and click <b>OK</b>. The <b>Create New Package Profile</b> dialog box opens.</li> <li>4. On the <b>Create New Package Profile</b> dialog box, enter a package profile name and a description (optional).</li> <li>5. Select the <b>Save paths for file names</b> check box to save the file path for your package profile. If you do not select the <b>Save paths for file names</b> check box, the default packaging directory specified on the Design Manager <b>Options</b> dialog is used.</li> </ol> <p>For more information about the Design Manager <b>Options</b> dialog box, see <i>Getting Started</i> in the Exstream Design and Production documentation.</p>
Delete a package profile	<ol style="list-style-type: none"> <li>1. Select the package profile that you want to delete.</li> <li>2. Click  .</li> </ol>
Edit a package profile	<ol style="list-style-type: none"> <li>1. Select the package profile that you want to edit.</li> <li>2. Click  . The <b>Edit Package Profile</b> dialog box opens.</li> <li>3. Make adjustments to the application's packaging settings as needed.</li> <li>4. Click <b>OK</b>.</li> </ol>
Create a package file	<ol style="list-style-type: none"> <li>1. Select the package profile that you want to use to create the package file.</li> <li>2. Click  . The <b>Build Package</b> dialog box opens displaying the packaging options that define the application's package profile.</li> <li>3. Make adjustments to the application's packaging settings as needed.</li> <li>4. Click <b>OK</b>.</li> </ol>

## 3.2.6 Packaging for Previews

You must create an archived package file of your application if you want to use one of the following features in Designer to preview a design:

- **Preview in External Viewer**—This feature lets you quickly preview any design page in a browser or other external viewer.
- **Device Preview**—This feature lets you simulate within Designer how an HTML (email) or HTML (web) container design will appear on current popular models of phones and tablets.

Before you can use either of these previewing features, you must first create an archived package file of your application that the engine can use to generate the preview. Similar to the way Design Manager or the Exstream production engine uses a package file to produce output, the previewing features use a page-level package file combined with an application-level archived package file to produce output for previewing.

For more information about using the Preview in External Viewer feature or the Device Preview feature in Designer, see *Designing Customer Communications* in the Exstream Design and Production documentation.

There are several considerations to keep in mind for managing the archived package files that are created in your design database. For example, you can use the archived package management tool in Design Manager to repackage, export, delete, or view the file size and the status of archived package files.

It is also important to keep in mind some considerations for archived package files when you perform certain database operations, such as when you use the Database Administrator utility (`DBAdmin.exe`) or when you load system settings from an XOB file.

Additionally, keep in mind that each time that you create a new package file using a package profile—regardless of whether in Design Manager or with the `Packager.exe` executable from the command line—the package file will be automatically archived in the directory that you specified for archived package files.

This section discusses the following topics:



- [“Managing Archived Package Files” below](#)
- [“Considerations for Archived Package Files When Performing Database Operations” on page 52](#)



## Managing Archived Package Files

If your design database includes multiple archived package files, you can use the archived package file management tool in Design Manager to view the file size and the status of those archived package files and perform file management tasks. The status of an archived package file is considered to be "current" unless the application, the components of the application (for example, output queue objects, language objects, or variables), or the package profile have been changed since the archived package file was created.

File management tasks include repackaging, exporting, or deleting archived package files. For example, you can repackage an archived package file to create a current copy in order to reduce the time that is required to generate a preview. Additionally, if you want to save a copy of an archived package file, you can export the file to a specified location. You can also use the archived package management tool to delete archived package files.

To manage archived package files in Design Manager, complete the following tasks as needed:

To	Do this
View your list of archived package files	<ol style="list-style-type: none"> <li>From the Menu bar, select <b>Tools &gt; Package archive management...</b> The <b>Archived Package File Management</b> dialog box opens and displays the list of archived package files.</li> <li>To close the <b>Archived Package File Management</b> dialog box, click <b>Exit</b>.</li> </ol>
View the status of archived package files	<ol style="list-style-type: none"> <li>From the Menu bar, select <b>Tools &gt; Package archive management...</b> The <b>Archived Package File Management</b> dialog box opens and displays the list of archived package files.</li> <li>Click <b>Show package status</b>. If the status of an archived package file is current, <b>Yes</b> appears in the <b>Current</b> column.</li> <li>To close the <b>Archived Package File Management</b> dialog box, click <b>Exit</b>.</li> </ol>
Update an archived package file	<ol style="list-style-type: none"> <li>From the Menu bar, select <b>Tools &gt; Package archive management...</b> The <b>Archived Package File Management</b> dialog box opens and displays the list of archived package files.</li> <li>In the <b>Archived package files</b> list, select one or more archived package files that you want to update.</li> <li>Click . The <b>Packager .exe</b> production executable packages the application, and then copies the package file to the specified path for archived package files. For more information about specifying a path for archived package files, see <i>Designing Customer Communications</i> in the Exstream Design and Production documentation.</li> <li>To close the <b>Archived Package File Management</b> dialog box, click <b>Exit</b>.</li> </ol>
Delete an archived package file	<ol style="list-style-type: none"> <li>From the Menu bar, select <b>Tools &gt; Package archive management...</b> The <b>Archived Package File Management</b> dialog box opens and displays the list of archived package files.</li> <li>In the <b>Archived package files</b> list, select one or more archived package files that you want to delete.</li> <li>Click .</li> <li>To confirm that you want to delete the selected archived package file(s), click <b>Yes</b>. The archived package file(s) no longer appears in the <b>Archived package files</b> list.</li> <li>To close the <b>Archived Package File Management</b> dialog box, click <b>Exit</b>.</li> </ol>

To	Do this
Export a copy of an archived package file	<ol style="list-style-type: none"> <li>From the Menu bar, select <b>Tools &gt; Package archive management...</b> The <b>Archived Package File Management</b> dialog box opens and displays the list of archived package files.</li> <li>Do one of the following: <ol style="list-style-type: none"> <li>In the <b>Archived package files</b> list, select an archived package file that you want to copy. <ol style="list-style-type: none"> <li>Click  .</li> <li>The <b>Export Package File</b> dialog box opens.</li> <li>In the <b>File name:</b> box, enter a name for the copy of the package file.</li> <li>To specify the location where you want to copy the file, use the <b>Export Package File</b> dialog box to browse to the location where you want to copy the file.</li> <li>Click <b>Save</b>. The <b>Export Package File</b> dialog box closes, and a copy of the package file is saved in the directory that you specified.</li> </ol> </li> <li>In the <b>Archived package files</b> list, select multiple archived package files that you want to copy. <ol style="list-style-type: none"> <li>Click  .</li> <li>The <b>Browse For Folder</b> dialog box opens.</li> <li>To specify the location where you want to copy the files, use the <b>Browse For Folder</b> dialog box to browse to the location where you want to copy the files.</li> <li>Click <b>OK</b>. The <b>Browse For Folder</b> dialog box closes, and the files that you selected are copied to the directory that you specified.</li> </ol> </li> </ol> </li> <li>To close the <b>Archived Package File Management</b> dialog box, click <b>Exit</b>.</li> </ol>

## Considerations for Archived Package Files When Performing Database Operations

Exstream uses a universally unique identifier (UUID) to uniquely identify each design database. You can view the UUID for a design database in the system settings in Design Manager. When you create a new design database, or when you perform certain database operations, Design Manager automatically assigns a new UUID. Archived package files are saved in a sub-directory of the global path that you specify for archived package files, and the UUID is used as the sub-directory name.

For more information about specifying a path for archived package files, see *Designing Customer Communications* in the Exstream Design and Production documentation.

Keep in mind the following considerations when performing database operations:

- When you use the Database Administrator utility (`DBAdmin.exe`) to copy a design database, a new UUID is automatically created for the new database.

- When you use the Database Administrator utility (`DBAdmin.exe`) to drop a design database, you are prompted to either delete the archived package files from the design database or to keep them.

When you run `DBAdmin.exe` from the command prompt to drop a design database, and you use the `AUTODROP` switch, archived package files are automatically deleted unless you use the `KEEPARCHIVEDPACKAGES` argument to keep them.

- When you load system settings from an XOB file to a design database, a new UUID is automatically assigned to the database.

For more information about using an XOB file to move content between design databases, see *Installation and Upgrade Information* in the Exstream Design and Production documentation.

- If you duplicate a design database by restoring a backup to a new schema, and you want to use both the original database and the new database, you must manually assign a new UUID for the new database. To assign a new UUID for the new database, in Design Manager, click **Change** on the **Basic** tab of the system settings for the application.
- When you delete an application or a packaging profile, the archived package files that are associated with the application or the packaging profile are automatically deleted. However, keep in mind that if the archived package files are stored at a network location, and the network location is unavailable, the archived package files are not deleted and you do not receive a warning message in this scenario. To manually delete archived package files, use the archived package file management tool.

For more information about using the archived package file management tool, see [“Managing Archived Package Files” on page 50](#).

## 3.3 Packaging Applications From the Command Prompt

Packaging applications from the Windows command prompt offers you the ability to approach the packaging process without the use of the Design Manager user interface. Packaging applications from the Windows command prompt offers an automated version of packaging applications from Design Manager. Using the Windows command prompt, a developer can perform all of the necessary packaging tasks using packaging switches. Packaging switches, both required and optional, correspond with packaging options available on the Design Manager interface, including the print resource options available for both SBCS and DBCS applications.

For more information about packaging options in the Design Manager interface, see [“Packaging Applications in Design Manager” on page 34](#).

You can package using a control file or directly from the command prompt. To package from the command prompt in Windows, use the following command:

```
Packager -APPLICATION="Application Name" <Other Switches>
```

On the command line, you must first call the executable ( `Packager.exe`). You must then use the `APPLICATION` packaging switch to specify the application you want to package. If the application name has spaces, you must use double quotation marks or the name is truncated.

Additional packaging switches required include the following:

- `DSN`
- `EXSTREAMUSER`
- `EXSTREAMPASSWORD`

If you want to use the packaging settings saved in an application's package profile, you can add the `PACKAGEPROFILE` packaging switch, which sets the package file options based on the specified profile. You must make sure that you first specify the `APPLICATION` switch before adding the `PACKAGEPROFILE` switch.

For more information about the `PACKAGEPROFILE` packaging switch, see [“Packaging switches” on page 59](#).

For more information about package profiles, see [“Managing Package Profiles” on page 48](#).

You can add optional packaging switches to increase packaging performance and lessen the potential for errors. A number of packaging switches contain dependencies on the values that you set for the `PACKAGETYPE` packaging switch. The `PACKAGETYPE` switch is an optional packaging switch that indicates the packaging resources that you want to include in the application package file.

When your application package file requires you to include a value for the `PACKAGETYPE` packaging switch, you must make sure that the placement of the `PACKAGETYPE` switch precedes all other optional packaging switches that are listed on the command line.

For more information about the `PACKAGETYPE` switch and other packaging switches, see [“Packaging switches” on page 59](#).

When your application production goals require you to include a sub-package of non-design or environmental objects, you must include additional commands to ensure the specified objects in the sub-package append to the main package file. Use the following packaging switches when you want to include a sub-package:

- **`PKGOBJECT`**—Specifies the non-design or environmental object in the sub-package file
- **`PKGOBJECTFILE`**—Specifies a file that contains the non-design or environmental objects in the sub-package file

You can include the `PKGOBJECT` and `PKGOBJECTFILE` switches as many times as is necessary on the command line.

For more information about the `PKGOBJECT` and `PKGOBJECTFILE` packaging switches, see [“Packaging switches” on page 59](#).

When packaging from the command prompt, make sure you correctly define all data files and output queues. Incorrect paths result in an error message and you cannot use the package file.

For more information about packaging using a control file, see [“Increasing Productivity with a Reusable Packaging Format” on page 71](#).

To package applications from the command prompt, complete the following tasks as needed:

- [“Performing Multiple Packaging Operations From the Command Line” below](#)
- [“Using Packaging Switches to Create Campaign and Document Package Files” below](#)

### 3.3.1 Performing Multiple Packaging Operations From the Command Line

You can run your packaging operations without a system administrator or developer monitoring the production runs, thereby maximizing your productivity and reducing operating costs. Exstream lets you perform multiple packaging operations from the command line by making multiple calls to the production executable ( `Packager.exe`) within a single batch ( `.bat`) file. When you use the `Packager.exe` production executable, there is no need to manually navigate messages. For example, an organization can implement the Exstream Design and Production platform as its document automation solution, even if the organization's existing production processes require it to produce multiple runs. And, calling the `Packager` executable can be done without the need for an employee to monitor the progress of each run.

To create a package file using `Packager.exe`, complete the following steps:

1. Call `Packager.exe` as needed.
2. Specify one application per call.
3. Include all required and optional packaging switches necessary for each application.

### 3.3.2 Using Packaging Switches to Create Campaign and Document Package Files

If you use command line packaging, you can use packaging switches to create campaign and document package files. You can package single campaigns and documents, or you can package all the campaigns and documents in a single folder.

When creating packages, you must always specify the application in which the documents or campaigns should be linked. From the command prompt, use the `CAMPAIGN` or `DOCUMENT` packaging switches to specify the folder path location of the campaign or document you want to target.

To use packaging switches to create campaign and document package files, complete the following tasks:

To	Do this
Create single document package or campaign packages	Insert one of the following switches in the command line: <ul style="list-style-type: none"><li>• -CAMPAIGN = CampaignName[, FolderPath]</li><li>• -DOCUMENT = DocumentName[, FolderPath]</li></ul>
Create a batch package for all the documents or campaigns in a single folder	Insert one of the following switches in the command line: <ul style="list-style-type: none"><li>• -CAMPAIGN = *, FolderPath [, EXCLUDEDOK]</li><li>• -DOCUMENT = *, FolderPath [, EXCLUDEDOK]</li></ul> <p>Batch packaging documents or campaigns is useful if, for example, you have several marketing design locations but store all the finished campaigns in a single folder.</p>

**Note:** These switches do not include any campaigns or documents included in subfolders within the folder you specify.

Use the EXCLUDEDOK parameter if you use other switches, such as the WIP switch, that limit the object versions that are packaged. This parameter changes the normally severe error to an informational one and ensures a package is created.

If all your campaigns or documents appear in the root folder, specify `_Exstream` as part of the file path. However, if the campaigns or documents are in a subfolder, you do not need to include the root folder. Instead, you specify the folder and any subfolders, separating them with a forward slash (for example, `Sample/Demonstration`).

### 3.3.3 Uploading a package file to the CAS repository from the command prompt

The common asset service (CAS) provides a centralized versioned repository for storing package files and allows a user to upload multiple versions of a package file along with associated metadata. You can upload your package files to the CAS when packaging from the command prompt.

Uploading your package files to the CAS also allows you to select and use the stored package files in the integrated Exstream platform environment and take advantage of the production engine and output delivery orchestration features. You can use the stored package files in Communications Builder and run the Exstream production engine from Control Center.

For complete information about using Exstream package files in Communications Builder, see *Creating Exstream Engine plug-ins* in *OpenText Exstream: Communications Builder Configuration Guide*. For complete information about setting up a Communications Server application and running the Exstream production engine from Control Center, see *Setting up Communications Server applications* in *OpenText Exstream: Communications Server Administration Guide*.



When you package from the command prompt, you must use the following switches to upload the package file to the CAS repository:

Switch	Value
UPLOAD_PACKAGE_TO_CAS	This packaging switch lets you upload the package file to the CAS.
EXSTREAMUSER=<userName>	This packaging switch lets you specify your Exstream user name, which is used with OTDS authentication when connecting to the CAS repository. The specified user must be an OTDS user.
EXSTREAMPASSWORD=<password>	This packaging switch lets you specify your Exstream password, which is used with OTDS authentication when connecting to the CAS repository.

Use the following optional switches to specify the upload settings:

Switch	Value
CASPACKAGENAME=<packageName>	<p>This packaging switch specifies the name of the package file. You must specify a name with 255 or fewer characters. If the name has spaces in it, and if you are using the CASPACKAGENAME switch at the command prompt, you must enclose the argument value in double quotation marks (" "). If you do not use this switch, the name of the Exstream application is used.</p> <div><b>Tip:</b> You can upload multiple package files with the same name to the CAS. To avoid confusion, OpenText recommends that you use the CASPACKAGENAME switch to specify a unique name when you upload a new package file to the CAS.</div> <p><b>Example</b></p> <p>-CASPACKAGENAME=MyPackageName</p>
CASPACKAGEDESCRIPTION=<description>	<p>This packaging switch specifies the description of the package file in the CAS. You must specify a description with 255 or fewer characters. If the description has spaces in it, and if you are using the CASPACKAGEDESCRIPTION switch at the command prompt, you must enclose the argument value in double quotation marks (" ").</p> <p><b>Example</b></p> <p>-CASPACKAGEDESCRIPTION="Package file to produce customer output"</p>
CASPACKAGEUPLOADTIMEOUT=<seconds>	<p>This packaging switch specifies the number of seconds to wait for a package file to finish uploading to the CAS before the upload times out. The default is 600 (10 minutes).</p> <p><b>Example</b></p> <p>-CASPACKAGEUPLOADTIMEOUT=300</p>

Switch	Value
CASCREATENEWPACKAGE	<p>This packaging switch specifies that a new package file object should be created in the CAS for the package file that you are uploading, even if a package file object from the same application already exists in the CAS.</p> <p>If you do not use this switch, a new package file object is created only if a file from the same application does not already exist. If a file from the same application exists, that file is updated and the version number for the file is incremented.</p> <div> <p><b>Tip:</b> When you use the CASCREATENEWPACKAGE switch, you can upload multiple package files with the same name to the CAS, even if they are created from the same application. To avoid confusion, OpenText recommends that you use the CASPACKAGENAME switch to specify a unique name when you upload a new package file to the CAS.</p> </div>
CASSTOPLEVEL=<errorLevel>	<p>This packaging switch specifies the level of error that stops the package file from being uploaded to the CAS.</p> <p>Select from the following options:</p> <ul style="list-style-type: none"> <li>• SEVERE—The package will not be uploaded in the case of a severe error. This is the default.</li> <li>• ERROR—The package will not be uploaded in the case of an error.</li> <li>• WARNING—The package will not be uploaded in the case of a warning.</li> </ul> <p><b>Example</b></p> <p>-CASSTOPLEVEL=ERROR</p>

Use the following optional switches if you want to override the OTDS or management gateway connection settings specified on the **Integration** tab in **System Settings** in Design Manager:

Switch	Value
OTDSURL=<url>	<p>This packaging switch specifies the URL for the OTDS server that you are using for user authentication.</p> <div> <p><b>Note:</b> If you are using a multi-tenant system, make sure that you use the correct OTDS URL for the tenant to which you want to connect.</p> </div> <p><b>Example 1: Single-tenant or default tenant in a multi-tenant system</b></p> <p>-OTDSURL=https://sampleserver.example.com:8443</p> <p><b>Example 2: Non-default tenant in a multi-tenant system</b></p> <p>- OTDSURL=https://sampleserver.example.com:8443/otdstenant/yourtenant2</p>

Switch	Value
OTDSRESOURCEID=<identifier>	<p>This packaging switch specifies the OTDS resource identifier that is associated with the resource for Exstream, as it appears in the OTDS server that you are using for user authentication.</p> <div> <p><b>Note:</b> If you are using a multi-tenant system, make sure that you specify the correct resource identifier for the tenant to which you want to connect.</p> </div> <p><b>Example</b></p> <p>-OTDSRESOURCEID=abd87460-39d4-4f2c-b30d-f4a2ff83727a</p>
MGWURL=<url>	<p>This packaging switch specifies the URL for the management gateway server that you are using to connect to the CAS repository that contains your CAS resources.</p> <p><b>Example</b></p> <p>-MGWURL=https://sampleserver.example.com:28600</p>
MGWTENANT=<tenantName>	<p>This packaging switch specifies the management gateway tenant that corresponds to the CAS repository that contains your CAS resources.</p> <p><b>Example</b></p> <p>-MGWTENANT=yourTenant</p>
MGWAPPDOMAIN=<yourDomain>	<p>This packaging switch specifies the application domain for connecting to the CAS repository that contains your CAS resources. This must be a domain within the tenant that is specified on the <b>Integration</b> tab or in the MGWTENANT switch.</p> <div> <p><b>Note:</b> If you are using a multi-tenant system, make sure that you specify an application domain that is within the tenant to which you want to connect.</p> </div> <p><b>Example</b></p> <p>-MGWAPPDOMAIN=yourDomain</p>

### 3.3.4 Packaging switches

You can use packaging switches when you:

- Package an application from the Windows command prompt
- Run packager.exe and use a package control file
- Perform multiple packaging operations from the command prompt by way of a batch file

The following table provides information about the packaging switches that you can use with your Exstream applications:

## Packaging switches

Switch	Value
APPLICATION=AppName, [campaigns   documents   pkgobject],[folder name]	<p>This packaging switch specifies the name of the application to package, with the optional campaign, documents, pkgobject or folder name parameters to target for packaging. If a target is not specified, it defaults to APPLICATION. The folder name where the application is located is optional. The default is "_Exstream". This switch is required.</p> <div> <p><b>Tip:</b> If all your campaigns or documents appear in the root folder, specify _Exstream as part of the file path. However, if the campaigns or documents are in a subfolder, you do not need to include the root folder. Instead, you specify the folder and any subfolders, separating them with a backslash. For example: Sample/Demonstration.</p> </div>
APPLICATION_MODE=SBCS   DBCS	<p>This packaging switch specifies the application mode for the application being packaged.</p> <p>Select from the following options:</p> <ul style="list-style-type: none"> <li>• SBCS—Specifies the application being packaged contains SBCS objects.</li> <li>• DBCS—Specifies the application being packaged contains DBCS objects.</li> </ul>
BUILDFONTS=value	<p>This packaging switch specifies whether to include all fonts, only fonts with names, or only fonts without names.</p> <p>Select from the following options:</p> <ul style="list-style-type: none"> <li>• ALL</li> <li>• WITHNAMES</li> <li>• WITHOUTNAMES</li> </ul> <p>When packaging resources, the PACKAGETYPE switch must use RESOURCESONLY as the argument.</p>
BUILDIMAGES	<p>This packaging switch specifies whether or not to build images. There is no argument for this packaging switch.</p> <p>When packaging resources, the PACKAGETYPE switch must use RESOURCESONLY as the argument.</p>

### Packaging switches, continued

Switch	Value
CAMPAIGN= CampaignName [,FolderPath, EXCLUDEDOK]	<p>Packages a specific campaign. The package includes the specified campaign and the resources and queues defined for the application.</p> <p>To specify all campaigns in a folder, use * for the campaign name. Campaigns in subfolders are not included.</p> <p>The EXCLUDEDOK parameter is valid only if you package all the campaigns in a folder. Use the EXCLUDEDOK parameter if you use other switches, such as -WIP, that limit the object versions packaged. It changes the normally severe error to an information one and ensures a package is created.</p>
CASCREATENEWPACKAGE	<p>When uploading a package file to the common asset service (CAS), this packaging switch specifies that a new package file object should be created in the CAS for the package file that you are uploading, even if a package file object from the same application already exists in the CAS.</p> <p>If you do not use this switch, a new package file object is created only if a file from the same application does not already exist. If a file from the same application exists, that file is updated and the version number for the file is incremented.</p> <div><b>Tip:</b> When you use the CASCREATENEWPACKAGE switch, you can upload multiple package files with the same name to the CAS, even if they are created from the same application. To avoid confusion, OpenText recommends that you use the CASPACKAGENAME switch to specify a unique name when you upload a new package file to the CAS.</div>
CASPACKAGEDESCRIPTION=<description>	<p>When uploading a package file to the common asset service (CAS), this packaging switch specifies the description of the package file. You must specify a description with 255 or fewer characters. If the description has spaces in it, and if you are using the CASPACKAGEDESCRIPTION switch at the command prompt, you must enclose the argument value in double quotation marks (" ").</p>

## Packaging switches, continued

Switch	Value
CASPACKAGENAME=<packageName>	<p>When uploading a package file to the common asset service (CAS), this packaging switch specifies the name of the package file. You must specify a name with 255 or fewer characters. If the name has spaces in it, and if you are using the CASPACKAGENAME switch at the command prompt, you must enclose the argument value in double quotation marks (" "). If you do not use this switch, the name of the Exstream application is used.</p> <div> <b>Tip:</b> If you use the CASCREATENEWPACKAGE switch, you can upload multiple package files with the same name to the CAS. To avoid confusion, OpenText recommends that you specify a unique name. </div>
CASPACKAGEUPLOADTIMEOUT=<seconds>	<p>When uploading a package file to the common asset service (CAS), this packaging switch specifies the number of seconds to wait for a package file to finish uploading to the CAS before the upload times out. The default is 600 (10 minutes).</p>
CASSTOPLEVEL=<errorLevel>	<p>This packaging switch specifies the level of error that stops the package file from being uploaded to the common asset server (CAS).</p> <p>Select from the following options:</p> <ul style="list-style-type: none"> <li>• SEVERE—The package will not be uploaded in the case of a severe error. This is the default.</li> <li>• ERROR—The package will not be uploaded in the case of an error.</li> <li>• WARNING—The package will not be uploaded in the case of a warning.</li> </ul>
CHECK_CAS_IMG_UPDATES	<p>This packaging switch specifies whether to include information about updates to common asset service (CAS) image resources in the packaging message file. There is no argument for this packaging switch.</p>

## Packaging switches, continued

Switch	Value
COMPRESSPACKAGEFILE	<p>This packaging switch specifies that the package file should be compressed. Compression takes place at the end of the packaging process, and compresses the contents that are located between the package header and the AQS manifest. This switch is recommended for package files that will be uploaded to OpenText cloud applications.</p> <p>Keep in mind the following considerations when using package file compression:</p> <ul style="list-style-type: none"> <li>• The APPLICATION packaging switch must target an application for packaging.</li> <li>• If you use the PACKAGETYPE packaging switch, compression is supported only if all print resources are included (PACKAGETYPE=ALL).</li> <li>• Compressed package files can be decompressed only by an Exstream production engine that is installed and running in a Windows, Linux, or AIX environment.</li> <li>• Although you should always compress Design and Production package files that are used with OpenText cloud applications, package file compression can also be used to reduce the amount of resources required to store package files locally or to transmit them to other platform resources such as CAS.</li> <li>• Package files that contain mostly images might not achieve a high rate of compression, because images are already in a compressed format.</li> </ul>
CONTROLFILE=fully qualified path	<p>This packaging switch specifies the name of the control file to use for packaging. The fully-qualified path you provide specifies the location of the control file you want to use for packaging.</p>
DBAUTHENTICATION=value	<p>This packaging switch specifies the database authentication mode:</p> <ul style="list-style-type: none"> <li>• <b>DEFAULT</b>—Uses the default Exstream user name and password</li> <li>• <b>WINDOWS</b>—Uses the Windows Authentication</li> <li>• <b>PROMPT</b>—Prompts for database user name and password</li> <li>• <b>EXSTREAM</b>—Exstream and database have the same user name and password</li> </ul>
DBPASSWORD=password	<p>This packaging switch specifies the password to use if it is not specified in the DSN. The DBPASSWORD packaging switch is required only if the database used in the application requires a password.</p>

### Packaging switches, continued

Switch	Value
DBSCHEMA=schema	This packaging switch specifies the database schema to use if it is not specified in the DSN. The DBSCHEMA packaging switch is required only if the database used in the application requires a schema.
DBUSER=user name	This packaging switch specifies the user name to use if it is not specified in the DSN. The DBUSER packaging switch is required only if the database used in the application requires a user name.
DEVICE=value	<p>Packages for a specific output object instead of the output queues in the application. If you do not specify a value for this switch, the engine packages the application for all output queues that are specified for the application.</p> <p>Select from the following options:</p> <ul style="list-style-type: none"> <li>• EXSTREAM</li> <li>• EXSTREAM_SBCS</li> <li>• EXSTREAM_DBCS</li> <li>• Printer name</li> </ul> <p>When in SBCS/DBCS mode, the EXSTREAM_SBCS and EXSTREAM_DBCS options specify the type of Exstream Viewer to which you want to package.</p> <p>For example:</p> <p>DEVICE=EXSTREAM_SBCS</p> <p>If you specify EXSTREAM, Exstream issues an error message.</p>
DOCUMENT= Document name [,Folder Path, EXCLUDEDOK]	<p>This packaging switch packages a specific document. The package includes the specified document and the resources and queues defined for the application.</p> <p>To specify all documents in a folder, use * for the document name. Documents in subfolders are not included.</p> <p>The EXCLUDEDOK parameter is valid only if you package all the campaigns in a folder. Use the EXCLUDEDOK parameter if you use other switches, such as -WIP, that limit the object versions packaged. It changes the normally severe error to an information one and ensures a package is created.</p>
DSN=dsn_name	This packaging switch specifies the name of the DSN to use. The DSN packaging switch is required.



### Packaging switches, continued

Switch	Value
EFFECTIVE=value	<p>This packaging switch specifies the effective date, or range of dates, for the package file. If it is not specified, the default is NOW.</p> <p>Select from the following options:</p> <ul style="list-style-type: none"> <li>• NOW</li> <li>• ASOF</li> <li>• RANGE</li> </ul>
ENDDATE=mm/dd/yyyy	<p>This packaging switch this switch is necessary only if ASOF or RANGE is specified for the EFFECTIVE switch. This specifies an "as of" date or the end date of a range.</p>
(DBCS) ENGINECONTROLFILE=file name	<p>This packaging switch lets you specify the control file engine uses during the pre-packaging step.</p> <p>During pre-packaging, the engine runs to gather information about the final engine run, such as what ranges of DBCS characters are used for a given font.</p>
EXCLUDE_APPROVED_MESSAGES_ WITH_UNAPPROVED_ TARGETING_ RULES	<p>When packaging for approved objects only, this packaging switch ensures that objects that use unapproved Library rules are not included in the package file. If an approved message or paragraph uses a Library rule that does not have an approved version, both the object and the rule are removed from the package file.</p> <p>When excluding approved messages or paragraphs with unapproved targeting rules, the WIP switch must use Approved as the argument.</p>
EXSTREAMPASSWORD=password	<p>This packaging switch lets you specify your Exstream password. The EXSTREAMPASSWORD packaging switch is required.</p>
EXSTREAMUSER=userName	<p>This packaging switch lets you specify your Exstream user name. The EXSTREAMUSER packaging switch is required.</p>
INCREMENTAL=value	<p>This packaging switch specifies whether or not to use incremental packaging. The default is NO.</p> <p>Select from the following options:</p> <ul style="list-style-type: none"> <li>• Yes</li> <li>• No</li> </ul>

## Packaging switches, continued

Switch	Value
JURISDICTIONMETHOD=value	<p>This packaging switch specifies which jurisdiction to include. The default is IGNORE.</p> <p>Select from the following options:</p> <ul style="list-style-type: none"> <li>• IGNORE</li> <li>• SINGLE</li> <li>• ALL</li> </ul>
JURISDICTIONEXCLUSIONREPORT	<p>This packaging switch specifies whether or not to include a report on excluded jurisdictions.</p>
JURISDICTIONNAME=name	<p>This packaging switch is only necessary if SINGLE is selected as the method in JURISDICTIONMETHOD. It lets you enter the name of the jurisdiction to include.</p>
MESSAGEFILE=	<p>This packaging switch lets you specify the fully-qualified file name and path to the location you want Packager to write the message file. If no value is specified, the default name of your message file is PackagerMessages.dat.</p>
MGWAPPDOMAIN=yourDomain	<p>When uploading a package file to the CAS repository, this packaging switch specifies the application domain for connecting to the CAS repository that contains your CAS resources. This must be a domain within the tenant that is specified on the <b>Integration</b> tab or in the MGWTENANT switch.</p> <div> <p><b>Caution:</b> Using this switch will override the application domain specified in the <b>Application domain</b> box on the <b>Integration</b> tab in <b>System Settings</b> in Design Manager.</p> </div>
MGWTENANT=tenantName	<p>When uploading a package file to the CAS repository, this packaging switch specifies the management gateway tenant that corresponds to the CAS repository that contains your CAS resources.</p> <div> <p><b>Caution:</b> Using this switch will override the management gateway tenant that is specified in the <b>Tenant name</b> box on the <b>Integration</b> tab in <b>System Settings</b> in Design Manager.</p> </div>

### Packaging switches, continued

Switch	Value
MGWURL=url	<p>When uploading a package file to the CAS repository, this packaging switch specifies the URL for the management gateway server that you are using to connect to the CAS repository that contains your CAS resources.</p> <div> <b>Caution:</b> Using this switch will override the management gateway URL that is specified in the <b>Management Gateway URL</b> box on the <b>Integration</b> tab in <b>System Settings</b> in Design Manager.         </div>
OTDSRESOURCEID=identifier	<p>This packaging switch specifies the OTDS resource identifier that is associated with the resource for Exstream, as it appears in the OTDS server that you are using for user authentication.</p> <div> <b>Caution:</b> Using this switch will override the OTDS resource identifier that is specified in the <b>Resource ID</b> box on the <b>Integration</b> tab in <b>System Settings</b> in Design Manager.         </div>
OTDSURL=url	<p>This packaging switch specifies the URL for the OTDS server that you are using for user authentication.</p> <div> <b>Caution:</b> Using this switch will override the OTDS URL that is specified in the <b>OTDS base URL</b> box on the <b>Integration</b> tab in <b>System Settings</b> in Design Manager.         </div>
PACKAGEFILE=file name	<p>This packaging switch specifies the name of the package file to create. The PACKAGEFILE packaging switch is required.</p>
PACKAGEPROFILE=ProfileName	<p>This packaging switch configures packaging settings based on the package profile. Packaging settings defined in the package profile override settings specified by packaging switches prior to the PACKAGEPROFILE switch.</p> <p>You must specify the APPLICATION packaging switch before you specify a package profile with the PACKAGEPROFILE switch.</p>

## Packaging switches, continued

Switch	Value
PACKAGETYPE=value	<p>This packaging switch specifies what to include in the package file. The default is ALL. When packaging resources, the PACKAGETYPE switch must use RESOURCEONLY as the argument.</p> <p>If you include the PACKAGETYPE switch on the command line or in your control file, you must make sure the switch precedes all of the other optional packaging switches that you use.</p> <p>Select from the following options:</p> <ul style="list-style-type: none"> <li>• ALL—Includes all print resources files in the application</li> <li>• RESOURCEONLY—Includes only print resource files in the application</li> <li>• WITHOUTRESOURCES—Does not include print resource files</li> </ul>
PKGOBJECT=object type:object name[:folder name]  object type:object OI[:folder name]	<p>This packaging switch packages a non-design or environmental object by type and name or object identifier into a sub-package.</p> <p>For example:</p> <p>PKGOBJECT=Barcode:MyBarcode[:MyFolder]</p> <p>The syntax is as follows:</p> <p>&lt;type&gt;:&lt;name[:folder name]   OI[:folder name]</p> <p>When you specify the PKGOBJECT packaging switch, the Packager.exe executable first attempts to identify the object name. If an object name is not identified, Packager.exe identifies the object identification number.</p>
PKGOBJECTFILE=name of text file	<p>This packaging switch specifies the location of the text file that contains a list of non-design and environmental objects that are packaged in a sub-package.</p> <p>For example:</p> <p>PKGOBJECTFILE=MyObjectFilePackage</p> <p>The syntax is as follows:</p> <p>&lt;type&gt;:&lt;name   OI&gt;[:folder name]</p> <p>In the text file, each object must be listed on a separate line.</p>

## Packaging switches, continued

Switch	Value
PKGSTYLESHEETMODE=value	<p>This packaging switch controls how Design Manager reapplies style sheet styles during packaging and when viewing objects in Designer if the styles are not consistent between WIP and approved objects. Possible values include the following:</p> <ul style="list-style-type: none"> <li>• 0—Do not reapply styles to drawn components on approved objects, except library components. 0 is the default.</li> <li>• 1—Always reapply styles to all drawn and library components, regardless of the approval state of the object.</li> <li>• 2—Do not reapply styles to drawn components on approved text messages or paragraphs, but always reapply styles to all other drawn and library components.</li> </ul> <div> <p><b>Note:</b> You must use the same value for the PKGSTYLESHEETMODE switch in both Design Manager and Designer.</p> </div>
REGISTER=value	<p>This packaging switch registers the application as an official package to track the last package date and to use other features that require the package file to be recognized. The default is NO.</p> <p>Select from the following options:</p> <ul style="list-style-type: none"> <li>• YES</li> <li>• NO</li> </ul> <div> <p><b>Caution:</b> Use the REGISTER packaging switch to register batch packages only when producing a package for production. Severe errors during packaging cause the package not to be registered.</p> </div>
RESOURCEPATH=filepath	<p>This packaging switch specifies the file path of the resource file.</p> <p>When you use the RESOURCEPATH packaging switch with the RESOURCEPACKAGEMETHOD packaging switch, you must keep the following in mind:</p> <ul style="list-style-type: none"> <li>• If RESOURCEPACKAGEMETHOD=ALLINONE, the RESOURCEPATH packaging switch takes the value of the resource file.</li> <li>• If RESOURCEPACKAGEMETHOD=ONEPERFILE, or RESOURCEPACKAGEMETHOD=RESOURCEPERFILE, the RESOURCEPATH packaging switch takes the value of the designated directory where you want to store the individual resources.</li> </ul> <p>When packaging resources, the PACKAGETYPE switch must use RESOURCESONLY as the argument.</p>

## Packaging switches, continued

Switch	Value
RESOURCEPACKAGEMETHOD=value	<p>This packaging switch specifies how to include the resources.</p> <p>Select from the following options:</p> <ul style="list-style-type: none"> <li>• ALLINONE</li> <li>• ONEPERFILE</li> <li>• RESOURCEPERFILE</li> </ul> <p>When packaging resources, the PACKAGETYPE switch must use RESOURCESONLY as the argument.</p>
RESOURCEREPORT=filepath	<p>This packaging switch specifies the file path for the resource report.</p> <p>When packaging resources, the PACKAGETYPE switch must use RESOURCESONLY as the argument.</p>
RUNID=boiLo[, boiHi]	<p>This packaging switch specifies the big object id (boi) of the TN_PACKAGING_RUN record for background packaging.</p>
STARTDATE=mm/dd/yyyy	<p>This packaging switch specifies the start date for effectivity. This switch is only necessary if you selected RANGE for EFFECTIVE.</p>
UPDATEEXISTING	<p>This packaging switch specifies whether or not to update an existing package file. You use this packaging switch when you want to enforce incremental packaging.</p>
UPDATEARCHIVED	<p>This packaging switch updates all of the archived package files, and consolidates the log files for all of the package files.</p> <p>This switch is useful if, for example, you want to schedule a recurring task to update the archived package files that are out of date. A package file is considered to be out of date if the application design includes changes since the last time that the archived package file was updated.</p> <p>Include the UPDATEARCHIVED switch in your control file.</p>
UPLOAD_PACKAGE_TO_CAS	<p>This packaging switch lets you upload the package file to the CAS.</p>
VERSIONCUSTOM=name	<p>This packaging switch specifies the name of the custom approval state created in the application. This packaging switch enforces the value for a custom approval state.</p>
VERSIONLABEL=name	<p>This packaging switch specifies the name of the custom label created in the application.</p>

#### Packaging switches, continued

Switch	Value
WIP=value	<p>This packaging switch specifies the status of objects to include in the package file. The default is ALL.</p> <p>If you package for WIP and there is no WIP object specified, then the package uses the <b>Version status</b> of the object that is next in line on the <b>Build Package</b> dialog box. In the case of a WIP package, the status can be (in order) either <b>Submitted for Approval</b> or <b>Approved</b>.</p> <p>Select from the following options:</p> <ul style="list-style-type: none"><li>• ALL</li><li>• SPECIFIED</li><li>• WIP</li><li>• WIPFORUSER</li><li>• SUBMITTED</li><li>• SUBMITTEDFORUSER</li><li>• APPROVED</li><li>• QUICKFIX</li></ul> <p>The QUICKFIX value has the same effect as selecting <b>Approved</b> as the <b>Version status</b> on the <b>Build Package</b> dialog box and then selecting the <b>Substitute Quick Fix versions</b> check box.</p> <p>For information about Quick Fixes, see <i>System Administration</i> in the Exstream Design and Production documentation.</p>
WIPUSER=user name	<p>This packaging switch specifies the user if SPECIFIED, WIPFORUSER, or SUBMITTEDFORUSER is selected as a value for WIP.</p> <p>If a user name is not specified for this switch, then the engine uses the last user that modified an object as the user name for WIPUSER at run time.</p>
WRITEPS_PDF FONTS	<p>This packaging switch lets you load Type 1 fonts onto an output device.</p>

## 3.4 Increasing Productivity with a Reusable Packaging Format

Using a package control file increases your productivity by utilizing a reusable packaging format. A package control file is similar to an engine control file. It contains all the switches you need when you package. Packager reads from the command prompt before it reads switches in

the control file. The switches you specify in the control file override the commands specified on the command line.

For information about packaging switches, see [“Packaging switches” on page 59](#).

To package an application using a control file:

1. Open a text editor program, such as Windows Notepad.
2. On the first line, enter the APPLICATION switch and specify the name of the application to be packaged.
3. Specify any additional switches to include in the control file, each on their own line. You must include the following required packaging switches:
  - DSN
  - EXSTREAMUSER
  - EXSTREAMPASSWORD
4. If you specify a value for the PACKAGETYPE packaging switch, you must make sure to list the PACKAGETYPE switch before all other optional packaging switches that are listed in your control file.

For more information about the PACKAGETYPE packaging switch, see [“Packaging switches” on page 59](#).

5. If desired, you can add comments to the packaging control file by specifying a \* or % symbol at the beginning of the comment.
6. Save the file.
7. At the command prompt, enter the following:  

```
Packager -CONTROLFILE="<Control file location>"
```
8. Press `ENTER` to execute the packaging command.

## 3.5 Integrating Changes at the Application Level

Because organizations must make changes to specific parts of their customer communications, Exstream offers you the ability to integrate changes at the application level through an easy repackaging process. For example, you must correct a variable that affects a campaign in the application. After an application has been packaged, you do not need to repackage the entire application unless changes are made at the application level. If changes are made only to campaigns or documents within an application, you can package the campaigns or documents separately using the **Target** drop-down list.



Consider the following when repackaging campaigns or documents within an application:

- If you use separate package files for your campaign(s) and application, you must repackage the campaign after you make changes to its properties. For example, after altering language layers or environment settings, you must repackage the campaign so your changes appear in the output.
- Some settings, such as **Submitted for Approval**, do not include Work in Progress objects, which is reflected in your output.


## 3.5.1 Creating Sub-Packages for Campaigns, Documents, and Objects

The ability to repackage at the sub-application level reduces costs and improves time to production. For example, if you must change the positioning of a design object in a document or the placement of a campaign teaser message, you do not have to spend the time and use many resources to open and change the whole application. You can simply target the exact object (for example, a single campaign or document, or non-design objects, such as barcodes) to make the change and repackage. You can repackage on the sub-application level in the same way you initially package an application.

### Creating a Single Campaign Sub-Package File

If you change only one campaign, you do not have to package all the campaigns in the application. You can create a sub-package that one campaign.

To create a single campaign sub-package, complete the following steps:

1. In the **Build Package** dialog box, select **Single Campaign** from the **Target** drop-down list. The **Campaign** box opens.
2. In the **Campaign** box, click  and then select the campaign you want to package.
3. Click **OK**.


A sub-package is created using the resources and queues defined for the application.

### Creating a Single Document Sub-Package File

If you change only one document, you do not have to package all the documents in the application. You can create sub-package that one document.

To create a single document sub-package, complete the following steps:

1. In the **Build Package** dialog box, select **Single Document** from the **Target** drop-down list. The **Document** box appears.

2. In the **Document** box, click  and then select the document you want to package.
3. Click **OK**.

A sub-package is created using the resources and queues defined for the application.

## Creating a Non-Design Object Sub-Package File


If you want to change or update a non-design object, you do not have to repackage the whole application. You can create a sub-package that contains only the objects you want to include.

To create a sub-package that contains non-design objects, complete the following steps:



1. In the **Build Package** dialog box, select **Non-Design Objects** from the **Target** drop-down list.

2. Next to the **Target** drop-down list, click .



The **Select Objects for Incremental Packaging** dialog box opens.

3. In the **Select Objects for Sub-Packaging** dialog box, click  to select the folder from which you want to include an object type.
4. In the **Select Objects for Sub-Packaging** dialog box, select the object you want to include in the sub-package from the **Type** drop-down list. Select from the following options:
  - **Barcode**
  - **Function**
  - **Live Action**
  - **Live Settings**
  - **Live Theme**
  - **Rule**
  - **Tag Set**
  - **Variable**
  - **View**

The selected objects appear in the **Available objects** area.

5. To include a single object from the list, click , or to include the list of objects in your sub-package, click .

The objects you select appear in the **Selected objects** area of the **Select Objects for Incremental Packaging** dialog box.

6. If you want to remove an object, click , or click .
7. Click **OK**.

## 3.6 Optimizing the Packaging Process Through Consolidating Applications

**Note:** Application consolidation is not currently supported in applications that are fulfilled through the OT2 platform using Exstream Document Generation on Demand.

In an organization with a rapidly growing customer base, or an organization that produces applications for a diverse array of customers, you can optimize the packaging process through application consolidation. You can consolidate the information from multiple package files (multiple applications) to produce a single output file.

Multiple package files are handled the same way as a single file. For multiple package files, the packager creates a separate index for each application. If you have licensed the Application Consolidator module, you can also create a primary application package file and multiple campaign or document sub-package files. All objects must match between the main package and sub-packages (for example, data files). For campaign or document sub-packages, the packager reads each sub-package file in the order listed. If an object in the sub-package is not in the index, it is added. If the object is already in the index, the sub-package object replaces the older object. Later sub-packages overwrite earlier sub-packages.

For more information about the Application Consolidator module, see *Creating Output* in the Exstream Design and Production documentation.

## 3.7 Implementing Collaborative Authoring Within the Packaging Process

Distributed database architecture allows implementation of collaborative, yet unconnected, authoring within the packaging process. Package files can be supplied by independent marketing companies without access to the organization's customer data files. A sample of data can be mapped in a dummy data file for testing and packaging.

For example, multiple marketing groups can incrementally package for only campaigns. These campaigns can be added to an existing package. When you package for only campaigns or documents, it creates a separate package file that can be run in batch mode with other packages.

For more information about campaigns and messages, see *Managing Marketing Messages* in the Exstream Design and Production documentation.

Distributed Database support is available if the system configuration includes the **Distributed Database ID** option on the **Workflow** tab.

To enable distributed databases, select the **Distributed Database ID** option on the **Workflow** tab of the **System Settings** in Design Manager.

For more information about system configuration, see *System Administration* in the Exstream Design and Production documentation.

### 3.7.1 Combining Files from Multiple Databases to Integrate Collaborative Authoring

When combining files from multiple databases to integrate collaborative authoring, you must make sure that all databases used in your organization have a unique database ID. The database ID is a number between 0 and 4280. Zero is reserved for the master database, which is usually the database with the majority of the application design information. "Child" databases, such as those used by a marketing department for campaign and message creation, or by a document creation center, are assigned a number other than zero. Failure to specify a unique database ID can result in collisions between objects from the different databases.

To combine files from multiple databases:

1. In the **System Settings**, click **System Configuration** on the **Basic** tab.  
The **System Configuration** dialog box opens.
2. On the **Workflow** tab, enter the database ID in the **Distributed database ID** box.

## 3.8 Managing Application Objects With Version Labels

You can create a library of version and effectivity packaging settings for a specific application. For each settings configuration, you provide a version label that describes the settings of the configuration. Then, when you package the application, you can select the version label associated with the packaging settings you want to use for a particular run. You can also use the label to see a list of all the objects that will qualify for the application and validate that those objects are the ones you want to include.

For example, suppose you are working with an application that might be run with three different version and effectivity settings to meet different requirements. In this case, you might create the following three version labels for the application:


- **Approved\_Now**—This label includes only objects in an approved state with an effectivity date that is valid on the day the package is created.
- **Approved\_2011**—This label includes only objects in an approved state with an effectivity date that makes objects valid after the year 2011.
- **WorkInProgress\_Now**—This label includes only objects in a work-in-progress state with an effectivity date that is valid on the day the package is created.

Before packaging, you can use the labels to view a list of the objects that will qualify for the application based on the effectivity and approval criteria. Then, when you package the application, you can simply select the configuration you want to use when you package rather than re-creating the packaging settings you want to use.

To use labels to pre-configure packaging settings, complete the following tasks as needed:


- [“Creating A New Version Label” below](#)
- [“Viewing the Objects in a Labeled Application” on page 79](#)
- [“Selecting a Version Label When Packaging an Application” on page 80](#)

### 3.8.1 Creating A New Version Label

1. In Design Manager, from the Library, drag the application for which you want to create version labels to the Property Panel.
2. Click the **Packaging** tab.
3. Click .

The **Build Label** dialog box opens.

4. In the **Label name** box, enter a name for the label. This name will appear on the **Build Package** dialog box if **Version label** is selected from the **Version method** drop-down list.
5. Use the other options on the drop-down list to specify how objects are selected for the application:

To	Do this
Select objects according to their version status	<ol style="list-style-type: none"> <li>From the <b>Version method</b> drop-down list, select <b>Version status</b>. The <b>Includes selected status and any statuses above it</b> slider area appears.</li> <li>Move the slider to the one of the following options: <ul style="list-style-type: none"> <li><b>Approved</b>—Includes all objects with a status of approved or archived.</li> <li><b>Submitted for approval</b>—Includes all objects with a status of submitted for approval, approved, and archived.</li> <li><b>Work in progress</b>—Includes all objects with a status of work in progress, submitted for approval, approved, and archived.</li> <li><b>Rejected</b>—Includes all objects with a status of rejected, work in progress, submitted for approval, approved, and archived.</li> </ul> </li> </ol>
Select objects from a particular user	<ol style="list-style-type: none"> <li>If you have selected <b>Submitted for Approval</b>, <b>Work in Progress</b>, or <b>Rejected</b> on the <b>Includes selected status and any statuses above it</b> slider, select the <b>Include versions from the following user</b> check box.</li> <li>Under the <b>Include versions from the following user</b> check box, click  . The <b>Select Design User</b> dialog box opens.</li> <li>From the list, select a design user.</li> <li>Click <b>OK</b>. The design user that you selected appears in the box under the <b>Include versions from the following user</b> check box. The user is only pertinent for Rejected, Work in Progress, and Submitted for Approval version statuses. Approved and Archived versions no longer have a user associated with them.</li> </ol>
Select objects with a particular custom approval state	<ol style="list-style-type: none"> <li>From the <b>Version method</b> drop-down list, select <b>Custom state</b>. The <b>Approval state</b> drop-down list appears.</li> <li>From the <b>Approval state</b> drop-down list, select a custom approval state.</li> </ol> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p><b>Note:</b> This option is available only if you have licensed the Advanced Design Workflow module and have created approval states.</p> </div>

6. Use the **Effective Date** options to specify how effectivity dates affect which objects can be selected for packaging:

To	Do this
Select the latest valid version of objects as of the current date	Select <b>As of Now</b> .

To	Do this
Select the latest valid version of objects as of a specified date	<ol style="list-style-type: none"> <li>Select <b>As of Date</b>.</li> <li>From the <b>To</b> drop-down list, select the date by which objects must be valid to be included.</li> <li>From the adjacent drop-down list, select an optional effective hour and minute.</li> </ol>
Select the valid version of objects as of a range of dates.	<ol style="list-style-type: none"> <li>Select <b>As of Date</b>.</li> <li>From the <b>From</b> drop-down list, select the earliest date by which objects must be valid to be included.</li> <li>From the <b>To</b> drop-down list, select the latest date by which objects must be valid to be included.</li> <li>From the adjacent drop-down list, select an optional effective hour and minute.</li> </ol>

- Click **OK**.


The **Build Label** dialog box closes and the label you created appears in the **Version Labels** area on the **Packaging** tab.

- Repeat step 2 through step 7 to create as many labels as needed.

## 3.8.2 Viewing the Objects in a Labeled Application

For each label associated with an application, you can view a list of the objects that will qualify for the application based on the criteria defined in the label. You can view the objects in a labeled application during testing to quickly see which objects will appear in the output. You can also view the objects to validate the label settings before going into production with an application.

To view the objects in a labeled application:

- In Design Manager, from the Library, drag the application for which you want to view the objects to the Property Panel.
- Click the **Packaging** tab.
- Select the label that controls the qualification criteria you want to check.
- Click  .

A list of all of the objects that will qualify for the application based on the version label settings appears in the Edit Panel. You can double-click the column headings to change the order in which the objects appear.

### 3.8.3 Selecting a Version Label When Packaging an Application

You can select a version label when packaging by selecting **Version label** from the **Version method** drop-down list.

For information about managing object versions in your package file, see [“Managing Application Objects With Version Labels” on page 76](#).



## Chapter 4: Building a Control File

A control file is a reusable format file that helps you increase productivity and save on costs. A control file lets you control the behavior of the engine by specifying required and optional engine switches in a centralized location. The format in which you write a control file varies depending on the platform you use. Any engine switch can be placed in a control file. When you use a control file, the only command necessary at the prompt is `CONTROLFILE=command`. Since you must run the production engine executable from the command prompt, options in the control file override options placed directly on the command line.

To build a control file, you must complete the following tasks:

1. [“Writing a Control File to Manage Application Production” below](#)
2. [“Validating Your License Key in the Control File to Avoid Engine Errors” on page 84](#)
3. [“Executing a Production Engine Run with a Control File” on page 86](#)

### 4.1 Writing a Control File to Manage Application Production

Writing a control file is the most optimal method you can use to manage application production. You can use a control file for packaging or for a production run. Enter the commands necessary for the production or packaging run into the control file. A package control file is created the same way as a control file for the engine. The only difference is that packaging switches are used instead of engine switches.

For more information about packaging with a control file, see [“Increasing Productivity with a Reusable Packaging Format” on page 71](#).

You enter switches into a control file using the same format that you use on the command line. All switches must have the following features:

- Preceded by a dash (-) or slash (/), depending on the platform
- In capital letters

**Caution:** If you include double quotation marks around a file name in a control file, You receive errors in the message file, that the specified file could not be found or created.

When you write a control file, you must select the engine switches necessary for the production run and add them to the file. The engine switches you use in your control file depends on the needs of your application and performance factors related to the engine.

To write an engine control file for use on the Windows, UNIX, or Linux platforms:

1. Open a text editor program, such as Windows Notepad.
2. Create a file for the control file. For example, CTRL1.txt. You can use any extension you prefer.
3. On the first line, enter the mandatory PACKAGEFILE switch with the argument specifying the package file you want to compose.
4. Specify any additional switches you want to include in the control file, listing each on its own line.

**Caution:** If you include double quotation marks ( `"` ) around a file name in a control file, you receive errors in the message file, that the specified file could not be found or created.

5. Save the file.

For a list of engine switches, see *Switch Reference* in the Exstream Design and Production documentation.

## 4.1.1 Writing Control Files for the z/OS Production Engine

Writing a control file for the z/OS production engine differs from that of a control file for the Windows or UNIX production engine. If you are operating the production engine on the z/OS platform and you want to write a control file, you must complete the following z/OS-specific tasks:

- [“Specifying the z/OS Name in the Control File” on the next page](#)
- [“Using a Fully Qualified Name in a z/OS Control File” on the next page](#)
- [“Using Rules to Specify a File Name in a z/OS Control File” on page 84](#)

The following example of a z/OS control file, used with dynamic images, can help you better conceptualize the examples used in the z/OS-specific tasks in this section.

```
-OUTPUTFILE=DD:EXOUTPUT  
-PACKAGEFILE=DD:PACKAGE  
-MESSAGEFILE=DD:EXMSGGS  
-REPORTFILE=DD:EXREPORT  
-RUNMODE=PRODUCTION
```

```
- TRACKIN=DISABLE  
- TRACKOUT=FILE  
- REPORT=CUSTOMER  
- ENIGNEDSN=P390LOC  
- IMPORTDIRECTORY=EXSTREAM.TIFF
```

This control file sets the import directory using the IMPORTDIRECTORY switch to EXSTREAM.TIFF.

**Note:** You must license the Dynamic Content Import module to use the IMPORTDIRECTORY engine switch.

## Specifying the z/OS Name in the Control File

To specify the z/OS name in the control file, you can use a combination of a Data Definition (DD) name and member name.

1. Set the DD to a Persistent Data Store (PDS) without specifying the member name.

For example:

```
//TIFFIMPO DD DSN=P390A.EXSTREAM.TIFF,DISP=SHR
```

2. Set the DD as the IMPORTDIRECTORY name and use the original data file.

For example:

```
-IMPORTDIRECTORY= DD:TIFFIMPO
```

**Note:** The z/OS engine is distributed with a control file. It has a built-in DD name:

```
DD:TIFFIMPO(SOUTHERN)
```

This is a valid z/OS file name and can help you get started.

## Using a Fully Qualified Name in a z/OS Control File

To use a fully qualified name, you must put single quotation marks around the entire name. If the full name is MYHLQ.EXSTREAM.TIFF (member), you must put the first quotation mark in the control file. For example: -IMPORTDIRECTORY= 'MYHLQ.EXSTREAM.TIFF

The trailing quote mark must be in the data file:

```
C Robert Stevens 123456 9/25/2000 (ASCII1)
```

```
T ABC (SOUTHERN)' (ASCII1)'  
T DEF (SOUTH)' (ASCII2)'
```

z/OS automatically appends the user's High Level Qualifier (HLQ) to the front of the data set. You do not need to use a fully qualified name because z/OS will automatically add the current HLQ.

## Using Rules to Specify a File Name in a z/OS Control File

You can also build the entire name using a rule to specify a file name (including the leading and trailing quote), where needed. In this case, you would not use the `-IMPORTDIRECTORY` switch at all. The following are options you can use to build the file name:

- Get the name of the fully qualified PDS from an initialization file.
- Get the member name only from the data file, but not parentheses or quotation marks.
- Build the fully qualified name using a rule by adding the following:
  - Leading quote
  - Open parenthesis
  - Trailing parenthesis and quotation marks

## 4.2 Validating Your License Key in the Control File to Avoid Engine Errors

To avoid production engine errors, you must validate your license key in the control file you use. Specifying the license key in your control file lets the engine confirm that all features used in the application are valid. If you use an expired key, or a key that does not carry all of the licensing features that appear in your application, then you receive error messages and the engine immediately stops processing.

### 4.2.1 Inserting the License Key on Windows and UNIX Platforms

If you are using a package file with an expired license key, you must insert a valid license key into the control file or the engine does not compose the package file.

Insert the following switch in your control file:

```
-KEY=value
```

The value is the text string of your license key. This value overrides the license key in the package file. You can copy the key characters from your Exstream Key File (EKF) by double-clicking on the license key, which, in the Windows environment, opens the license key in a program of your choice. In other environments, you can open the file using a text editor.

`-KEY=value`

When you run the engine with the control file, the new license key is used with the old package file.

When you copy this license key into your control file it must be an exact match. To prevent errors, copy and paste the license key into the control file you are using.

## Using the CONTROLFILE Switch on Windows and UNIX Platforms

You can call other control files from within a control file. This option is available to make license key management easier. For example, you can have a control file using only the KEY switch. When you need to change the license key, change it in this control file and then you can link all your control files to the KEY control file.

### 4.2.2 Inserting the License Key on the z/OS Platform

In z/OS, you must use uppercase characters. However, the license key must consist of the original mixed case characters. For example:

Your key: `XxxxXxXXXxxxxXxXxXxXxXXxxXXxxxXXxxxxXxXxxxXXxxx`

The license key must be specified exactly like this in the control file.

If you edit your z/OS control file on z/OS, the characters you enter for the license key are converted to upper case. For example:

`-KEY=XX`

This is not a valid license key, as the key must be mixed case. To ensure your license key is mixed case, set the z/OS command CAPS OFF. If you do not use this command, all text is automatically converted to uppercase.

## Using the KEYPART Switch with Large Keys on the z/OS Platform

On z/OS systems you can use the KEYPART switch to split a key string into multiple parts. The KEYPART switch is used for keys that are too large to fit on a single line of Job Control Language (JCL). When the engine processes multiple KEYPART switches, it combines the parts in the order supplied to form a single key string.

## 4.3 Specifying Multiple Package File Operations in the Control File

You must follow a specific syntax to ensure you specify multiple package file operations in the control file. In normal engine processing, or in step one for output sorting, all the packages for an application can be specified in a list in the control file.

The application package file must be specified first. For example:

```
-packagefile=App_master.pub  
-packagefile=4_4_2002__Doc.pub  
-packagefile=New_Phase_4_Camp_and_docs.pub  
-packagefile=One_changed_doc_and_one_new_doc.pub
```

You can also specify a text file containing the preceding list. For example:

```
-APP_PACKAGEFILES=MyPackageList.txt
```

In Post-Sort mode, available with the High-Volume Delivery module only, package files can be specified in the same manner, but if you are using the Application Consolidator module to consolidate more than one application, you must use one APP\_PACKAGEFILES switch per application.

For more information about output sorting, Post-Sort mode, and the Application Consolidator module, see *Creating Output* in the Exstream Design and Production documentation.

## 4.4 Executing a Production Engine Run with a Control File

To execute an engine run with a control file, you must enter the CONTROLFILE switch on the command line to specify the location of your control file.

To run the engine using a control file:

1. Working from the Exstream executable's directory, enter one of the following commands to start the engine run from the command prompt:
  - In Windows, the command is `prodengine`.
  - In UNIX, the command is `Engine`.
2. Enter a space after the `Engine` or `prodengine` command.

3. After the space, enter the `CONTROLFILE` command to specify the location of the control file.
4. Press `ENTER` to start the engine run.

For more information about executing production engine runs, see [“Running the Production Engine” on page 123](#).

# Chapter 5: Optimizing Production Engine Performance

From the moment a design user begins designing an application, each design element included in the application affects the production engine. For the engine to handle large amounts of variable data, rules, and other design elements, the application must adhere to some best practice specifications to limit the number of potential errors that might occur when the engine runs. In most cases, the testing phases of packaging verify that the production can efficiently run the package file. However, due to the complex and diverse nature of most applications, a developer running the production engine must utilize general and specific engine switches to maximize the processing speed and timing of the engine.

The engine must compose specific design elements at certain times so all information appears correctly on a page. How you create the design directly affects how the engine composes design objects in your application. Additionally, depending on the platform you use to run the production engine, you must consider the processing speed of the engine.

When you are ready to run the production engine, you should have all design elements in place and your application( s) in a package file ready to be processed by the engine for output. Since the engine must process multiple variables, rules, and other design elements, it is best to make sure you understand how the engine performs processing actions so you can optimize performance.

This chapter discusses the following topics:

- [“Maximizing Engine Processing Capabilities” below](#)
- [“Understanding Engine Processing and Timing to Improve Production” on page 91](#)
- [“Managing Production Engine Processing Memory” on page 113](#)

## 5.1 Maximizing Engine Processing Capabilities

One of the most important pieces in the implementation of Exstream solutions in your infrastructure is maximizing the processing capabilities of the production engine. Applications designed using only Exstream and those that integrate with the production capabilities of Exstream must be packaged using a package file (a file with the .pub extension). The package file is the most optimal format to house an application. The way the engine processes information is depends on how the application is designed. Throughout the design stages, a design user must select the properties that allow the document to be efficiently processed by the engine. For example, failure to set the appropriate late compose properties on an object can cause serious errors in your finished product.



The following concepts are important to understand before seeing each step the engine takes to optimally process information contained within a package file:

- How packaging results in optimal engine performance
- How the engine uses resource management
- How to use package messages with the engine
- How the engine uses the package file

Understanding these concepts and processes helps you avoid potentially devastating errors in producing your document solutions.

This section discusses the following topics:

- [“Understanding Application Packaging Concepts to Achieve Optimal Engine Performance” below](#)
- [“Understanding Resource Management Concepts to Achieve Optimal Engine Performance” on the next page](#)
- [“Addressing Packaging Messages to Analyze Engine Results” on the next page](#)
- [“How the Engine Uses the Package File to Determine Special Features and Allocate Memory” on page 91](#)

## 5.1.1 Understanding Application Packaging Concepts to Achieve Optimal Engine Performance

The following application packaging concepts can help you achieve optimal engine performance:

- All static design content is identified and overlays are created for the outputs that support them. The overlays are then reused during processing. You can create and include overlays in the print stream on the **Resource Management** tab of the output object in the Property Panel.
- The packaging process converts objects into a format devised for the engine. For example, version and history information is not relevant to the engine, so the packaging process strips out this information.
- Special functionality is identified as an on/off switch so the engine skips unnecessary processing. Packaging options are recognized and only the resources required for that engine run are used. For example, if your application does not require table sorting, the engine ignores this functionality.

For more information about packaging applications, see [“Packaging an Application” on page 32](#).

## 5.1.2 Understanding Resource Management Concepts to Achieve Optimal Engine Performance

Understanding resource management concepts and how the engine uses resource management can help you to achieve optimal engine performance. When you enable resource management properties in the design stages, the engine processes certain resources, such as images, fonts, and static content, in a specific way that prevents the engine from composing certain elements at an inappropriate time. The engine determines the composition of these items according to the application's properties. This means that a concise strategy, or best practice, must be in place while designing, which allows the engine to effectively process complex elements.

For more information about resource management, see *Creating Output* in the Exstream Design and Production documentation.

Consider the following design elements in relation to the engine:

- **Fonts**—For outputs that use raster or bitmap fonts, there is a font for each point size and rotation. For outline font technology, one font is used per font face.
- **Colors**—For PostScript drivers, the beginning of the PostScript file contains each named color in the package file. If the named color has an **Output Color Name**, the engine queries the printer for the calibrated CMYK values. If the value is known, the engine defines the color using the calibrated CMYK value. If the value is not known, the color is defined using the CMYK values entered in the design environment to define the named color, which may not match the values at the printer. The end of the PostScript file lists the named colors used.

For information about named colors, see *Designing Customer Communications* in the Exstream Design and Production documentation.

## 5.1.3 Addressing Packaging Messages to Analyze Engine Results

Make sure you read and address any packaging messages, including informational messages, as well as review and address engine messages so you can analyze engine results. Packaging messages, which appear as you package an application, can help explain non-intuitive engine results that are not documented in engine messages. Some packaging messages are contextual. That is, the context or use of the page determines the type of error checking performed. As a result, messages issued during packaging can be unique to the object. Other packaging messages are general to the application (for example, if a design font is replaced during packaging because it is not available).

A package file, engine, and message symbolics file comprise everything required to run the engine. You can use these elements repeatedly with different input data to produce different runs of an application.

### 5.1.4 How the Engine Uses the Package File to Determine Special Features and Allocate Memory

The engine reads the package file's index to determine special features, such as late compose, and to perform any required special setup, such as autofit text for charts. The engine reads an inventory of all objects and resources in the package file and builds an index in memory.

In Windows, UNIX, and Linux, the engine reads objects from the package file as necessary. The engine searches through the package file quickly and as a result, objects are loaded as needed. In addition, the engine unloads large objects, such as output resources, from memory after they are used. The index contains only the objects required for the current run.

On the z/OS platform, when package files are loaded, the engine tests them to verify if they are VSAM (Virtual Storage Access Method) package files. The results of the engine's tests appear in the message file header. If VSAM package files are loaded, the engine loads objects as necessary, similar to Windows and UNIX, to enhance performance. The index of available objects in memory contains only the objects required for the current run. However, if any of the package files are not VSAM, the engine reads all objects from all the package files into memory. The index is fully populated for the entire run of the engine.

## 5.2 Understanding Engine Processing and Timing to Improve Production

Understanding engine processing and timing helps you to improve the overall production of applications by showing you how the layout and order of your design can ultimately impact your output. Additionally, you can better identify and resolve unexpected results if you understand how the engine processes customer data.

Engine processing can differ depending on the format of the data in your customer driver file, and depending on whether you are using a two-pass engine process to complete the output sorting and bundling process. Depending on your application setup, use the following sections for additional information about the engine process and how the engine applies timing to your application:

If	Use this information
You are using a data file with a general layout or with data sections	<a href="#">"Engine Processing with General Data and Data Sections" on the next page</a>

If	Use this information
You are using a data file with a general layout or with data sections, and you are using two-pass engine processing for output sorting and bundling	<ul style="list-style-type: none"><li>• For pre-sort engine processing, see <a href="#">"Engine Processing with General Data and Data Sections"</a> below</li><li>• For post-sort engine processing, see <a href="#">"Post-Sort Engine Processing with General Data and Data Sections"</a> on page 99</li></ul>
You are using a schema model data file	<a href="#">"Engine Processing with Schema Model Data Files"</a> on page 103
You are using a schema model data file and you are using two-pass engine processing for output sorting and bundling	<ul style="list-style-type: none"><li>• For pre-sort engine processing, see <a href="#">"Engine Processing with Schema Model Data Files"</a> on page 103</li><li>• For post-sort engine processing, see <a href="#">"Post-Sort Engine Processing with Schema Model Data Files"</a> on page 109</li></ul>

## 5.2.1 Engine Processing with General Data and Data Sections

When your application uses a customer driver file that uses a simple data layout (such as general data) or a more complex data layout (such as data sections), use the following engine processing information to help you understand the engine processing order and engine timing.

For more information about customer driver files with general data or data sections, see *Using Data to Drive an Application* in the Exstream Design and Production documentation.

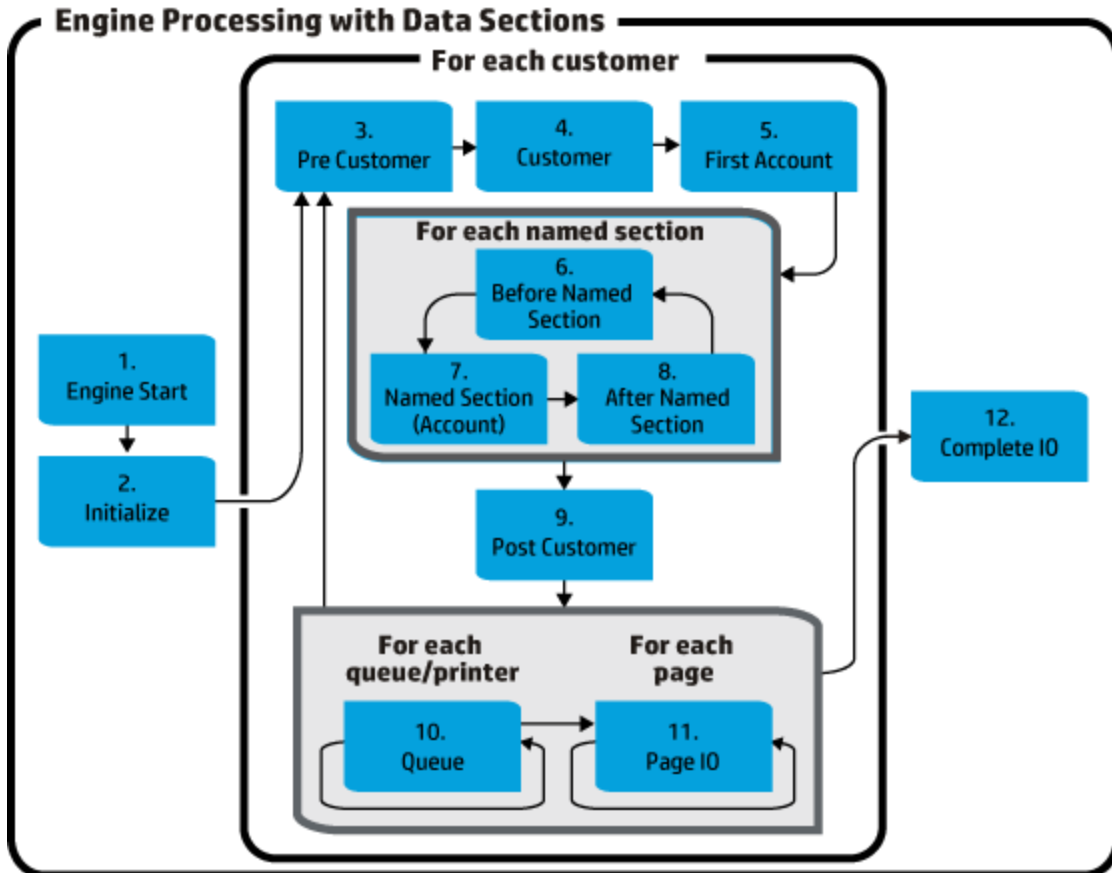
When you review engine processing for your application, keep in mind the following considerations:

- The engine actions are always completed in the same order, regardless of how you design an application. The differences in your output are determined by the way in which you build your application.
- Exstream Live applications require special timing considerations due to differences in variable substitution and rule execution timing. For Exstream Live applications, you can also use special Exstream Live-specific system variables. These system variables have unique behaviors that you should be aware of before you use them.

For more information about special timing considerations in Exstream Live and Exstream Live-specific system variables, see *Designing for LiveEditor* in the Exstream Design and Production documentation.

The following flowchart illustrates an engine processing flow with data files that contain a general data layout or data sections:

Engine processing flow with general data or data sections



The following table explains in detail what happens during each step in the previous flowchart, and when system variables are populated:

Engine processing steps and engine actions in an engine process with general data or section data

Step	Engine action	System variables populated
1. Engine Start	<ol style="list-style-type: none"> <li>1. Process command line arguments and control file.</li> <li>2. Read the package file.</li> <li>3. Process all initialization files.</li> </ol>	

Engine processing steps and engine actions in an engine process with general data or section data, continued

Step	Engine action	System variables populated
2. Initialize	<ol style="list-style-type: none"> <li>1. Reset system variables.</li> <li>2. Compute formulas.</li> <li>3. Write report files with IO time of <b>After initialization files read</b>.</li> </ol>	SYS_DateCurrent  SYS_VirtMsg_Contents  SYS_VirtMsg_Identifier  SYS_VirtMsg_Name  SYS_VirtMsg_Priority
Begin customer processing: The engine completes step 3 through step 11 for each customer		
3. Pre Customer	<ol style="list-style-type: none"> <li>1. Read records until engine reaches the first section or the start of the next customer.</li> <li>2. Set variables from customer records.</li> <li>3. Prepare campaign and message list by firing campaign and message rules (<b>After initial customer data</b>).</li> <li>4. Write report files with IO time of <b>After initial customer data</b>.</li> </ol>	SYS_CustomerBegByte SYS_CustomerBegRecord SYS_DriverFileOffsetBytes SYS_DriverFileOffsetBytesEnd SYS_DriverFileOffsetRecs SYS_DriverFileOffsetRecsEnd SYS_WeightStartGrams SYS_WeightStartOunces
4. Customer	<ol style="list-style-type: none"> <li>1. Set locale and languages.</li> <li>2. Read reference files by IO time.</li> <li>3. Compute formulas.</li> <li>4. Prepare document list firing document rules.</li> <li>5. Prepare page list firing page rules.</li> <li>6. Compose documents.</li> </ol>	SYS_CustomerEffectiveDate SYS_CustomerInRun SYS_CustomerJurisdiction SYS_DateAsOf (user defined) SYS_DDAAOutputUserData (user defined) SYS_DocumentNames SYS_LanguageCustomer SYS_LanguageDefault SYS_LocaleCustomer SYS_LocaleDefault SYS_MktgPagesAllowed SYS_PageInDocument SYS_SubDocInDocument SYS_WeightGrams SYS_WeightOunces
5. First Account		SYS_TableRow

Engine processing steps and engine actions in an engine process with general data or section data, continued

Step	Engine action	System variables populated
Begin named section processing: The engine completes step 6 through step 8 for each named section		
6. Before Named Section	<ol style="list-style-type: none"> <li>1. Read reference files by IO time.</li> <li>2. Set variables from record until next record marks a new section.</li> <li>3. Prepare campaign and message list for this section by firing campaign and message rules (<b>Each section, until qualified</b> and <b>Each section, sum qualified</b>).</li> </ol>	
7. Section (Account)	<ol style="list-style-type: none"> <li>1. Read reference files by IO time.</li> <li>2. Compute formulas.</li> <li>3. Prepare document list for section-based documents.</li> <li>4. Prepare page list firing page rules.</li> <li>5. Write report files with IO time <b>After each data section</b> and <b>After named data section</b>.</li> </ol>	
8. Post Named Section	<ol style="list-style-type: none"> <li>1. Write report files with IO time <b>After named data section</b>.</li> <li>2. Read records and set variables until the next record marks a new customer.</li> </ol>	

Engine processing steps and engine actions in an engine process with general data or section data, continued

Step	Engine action	System variables populated
9. Post Customer	<ol style="list-style-type: none"> <li>1. Prepare campaign and message list by firing campaign and message rules (<b>After all customer data</b>).</li> <li>2. Compute formulas.</li> <li>3. Prepare document list for campaigns firing document rules.</li> <li>4. Prepare page list firing rules.</li> <li>5. Write report files with IO time <b>At end of customer, before campaigns</b>.</li> <li>6. Order and weigh documents.</li> <li>7. Add static back pages and business (ordered) messages.</li> </ol>	<p>SYS_CampaignFirstPage</p> <p>SYS_CampaignPriority</p> <p>SYS_CampaignsQualified</p> <p>SYS_CampaignsSent</p> <p>SYS_SheetUsedByPaperTypes</p> <p>SYS_SheetTotalInDocument</p> <p>SYS_SheetTotalInRun (increments for each customer)</p> <p>SYS_TableRowOnPage</p> <p>SYS_TableRowPage</p> <p>SYS_TableRowTotal</p>
Begin queue processing: The engine completes the following engine actions for each queue that is qualified by a rule		
	<ol style="list-style-type: none"> <li>1. Add footnotes to current pages in document.</li> <li>2. Add marketing messages to current pages in documents.</li> <li>3. Add campaign-driven pages.</li> <li>4. Add any remaining inserts.</li> <li>5. Add blank back pages.</li> <li>6. Count pages and set page numbers.</li> <li>7. Process indexes and tables of content.</li> </ol>	



Engine processing steps and engine actions in an engine process with general data or section data, continued

Step	Engine action	System variables populated
Begin page and composition processing: The engine completes the following engine actions for all pages		
	<ol style="list-style-type: none"><li>1. Order and weigh again.</li><li>2. Add static back pages and business (ordered) messages.</li><li>3. Count pages and set page numbers.</li></ol>	
Continue queue processing: The engine completes step 10 through step 11 for each queue		

Engine processing steps and engine actions in an engine process with general data or section data, continued

Step	Engine action	System variables populated	
10. Queue	<ol style="list-style-type: none"> <li>1. Set queue system variables.</li> <li>2. Write report files with IO time of <b>Queue break and end of queue</b>.</li> <li>3. Run queue rules and perform final document selection per queue.</li> <li>4. Process all objects flagged as late compose (Last pass compose).</li> <li>5. Evaluate recipient data and update content as necessary for recipient copies.</li> <li>6. Sort pages in documents if required.</li> </ol> <p>While many variables can be used on the page with late object timing, step 10, or Queue, triggers their use (update or late composition)</p>	<div> <div>SYS_BreaksInQueue</div> <div>SYS_ByteInBreak</div> <div>SYS_ByteInQueue</div> <div>SYS_CurrentProfile</div> <div>SYS_CustomerChecksum</div> <div>SYS_DocPrintedValue</div> <div>SYS_DocumentChecksums</div> <div>SYS_DocumentInBreak</div> <div>SYS_DocumentInQueue</div> <div>SYS_DocumentTotalInRun (increments for each customer)</div> <div>SYS_InserterBinPriorities</div> <div>SYS_InserterBins</div> <div>SYS_InserterBinString</div> <div>SYS_InserterInQueue</div> <div>SYS_MktgPagesAdded</div> <div>SYS_MsgContents</div> <div>SYS_MsgIdentifier</div> <div>SYS_MsgName</div> <div>SYS_NumBannersInQueue</div> <div>SYS_PageFlows</div> <div>SYS_PageInBreak</div> <div>SYS_PageInQueue</div> <div>SYS_PagePhysicalInDocument</div> </div> <div> <div>SYS_PagePrintedValue</div> <div>SYS_PageStart</div> <div>SYS_PageTotalInDocument</div> <div>SYS_PageTotalInRun (increments for each customer)</div> <div>SYS_PageTotalPhysicalInDocument</div> <div>SYS_PageTotalPrinted</div> <div>SYS_PaperTypes (increments for each customer)</div> <div>SYS_PDLInQueue</div> <div>SYS_PrinterInQueue</div> <div>SYS_QueueCurrent</div> <div>SYS_QueueFileName</div> <div>SYS_QueuesToCustomer</div> <div>SYS_RecordInBreak</div> <div>SYS_RecordInQueue</div> <div>SYS_SectionName</div> <div>SYS_SheetInBreak</div> <div>SYS_SheetInDocument</div> <div>SYS_SheetInQueue</div> <div>SYS_SortIndex</div> <div>SYS_SubDocumentInBreak</div> <div>SYS_SubDocumentInQueue</div> <div>SYS_TotalSubDocsInDocument</div> <div>SYS_VolSerInQueue</div> </div>	
		<p>Variables that might be modified by other processes at later events:</p> <ul style="list-style-type: none"> <li>• SYS_DocumentNames</li> <li>• SYS_MktgPagesAllowed</li> <li>• SYS_PageInDocument</li> <li>• SYS_SubDocInDocument</li> <li>• SYS_WeightGrams</li> <li>• SYS_WeightOunces</li> </ul>	

Engine processing steps and engine actions in an engine process with general data or section data, continued

Step	Engine action	System variables populated
Continue page and composition processing: The engine completes the following step for each page		
11. Page IO	<ol style="list-style-type: none"><li>1. Compute formulas.</li><li>2. Write report files with IO time of <b>When each page is selected.</b></li><li>3. Output pages.</li></ol>	
12. Complete IO	Write report files with IO time of <b>Completion of all customers.</b>	

## 5.2.2 Post-Sort Engine Processing with General Data and Data Sections

**Note:** You must have licensed the Output Sorting and Bundling module to execute two-pass engine processing.

For information on the Output Sorting and Bundling module, see *Creating Output* in the Exstream Design and Production documentation.

In two-pass engine processing (separately known as pre-sort engine processing and post-sort engine processing), the engine is run twice in order to sort and bundle your output for postal discounts. During the first engine run (pre-sort engine processing), the engine produces customer data for the run that you can manipulate with third-party software to sort, cleanse, and update. After user sorting, you run the engine a second time (post-sort engine processing) in order to produce output according to the sorted information.

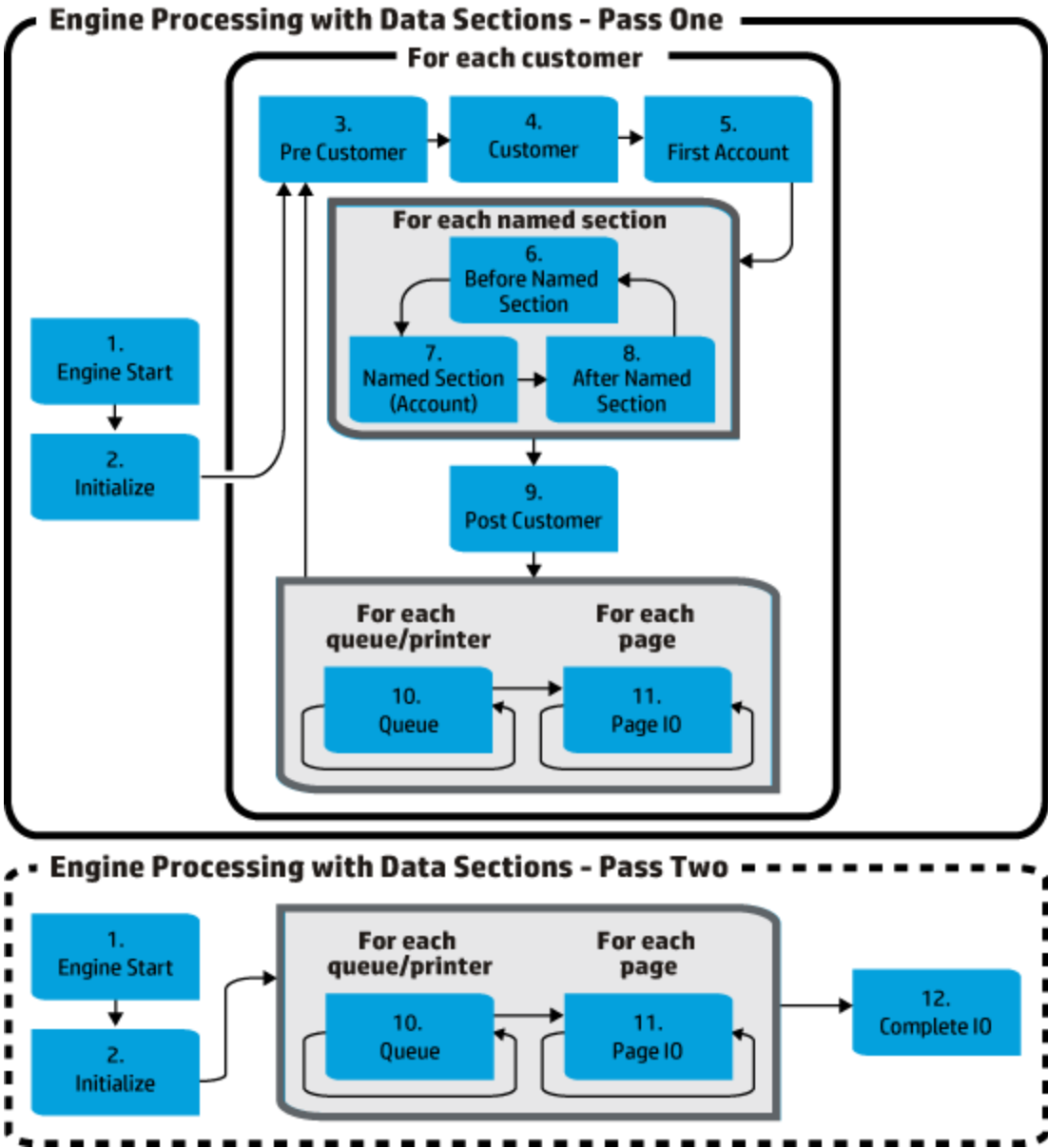
With two-pass engine processing, keep in mind the following considerations:

- All documents and pages are stored in pre-sort engine processing and re-read in post-sort engine processing.
- Only late compose objects are drawn in post-sort engine processing.
- Post-sort engine processing does not read the driver or reference data files, nor does it perform many of the events of pre-sort engine processing.
- Sort data file entries are a capture of the engine before the final late compose step is performed. Each entry contains the following:

- The drawn documents and pages
  - The design objects to be drawn in post-sort engine processing
  - Any variables that are used in post-sort engine processing
- 
- When each customer's sort data entry is read, all post-sort engine processing variables are reset to their pre-sort engine processing value. Simple tasks like user-defined counters will seem to work incorrectly. Thus, variables that are used in post-sort engine processing, but that are not to be restored to their pre-sort engine processing values for each entry must be flagged as such. There are various options for doing this, such as using a post-sort initialization file.

The following flowchart illustrates how engine processing differs when you use the two-pass engine process that is required for output sorting and bundling:

Pass Two Engine processing flow with general data or data sections



The following table explains how pre-sort engine processing differs from a general engine run when you use two-pass engine processing:

#### Pre-sort engine processing in a two-pass engine process

Step	Engine action
1-9	<p>In pre-sort engine processing for two-pass engine processes, the engine actions and system variables that are populated for step 1 through step 9 are identical to engine processing with general data and data sections.</p> <p>For more information about engine processing for general data and data sections, see <a href="#">"Engine Processing with General Data and Data Sections" on page 92</a>.</p>
10. Queue	<ol style="list-style-type: none"> <li>1. Set the queue system variables.</li> <li>2. Write report files with IO time of queue break and end of queue.</li> <li>3. Run queue rules and perform final document selection per queue.</li> <li>4. Last pass compose—process all objects flagged as late compose. Note that objects flagged as after document sorting or that use variables affected by document sorting (for example SYS_PageInDocument) are not composed yet.</li> <li>5. Evaluate recipient data and update content as necessary for recipient copies.</li> <li>6. Write customer and recipient entries to the sort index file.</li> </ol>
After all customers:	
12. Complete IO	Write report files with IO time of <b>Completion of all customers</b> .

At the end of pre-sort processing, you can sort the sort index file and change the fields that can be updated.

During post-sort engine processing, the engine produces output based on information that is sorted in pre-sort engine processing.

The following table explains in detail what happens during each step of the post-sort engine processing, as shown in the previous flowchart, and when system variables are populated:

#### Post-sort engine processing in a two-pass engine process

Step	Engine action	Variables populated
1. Engine Start	<ol style="list-style-type: none"> <li>1. Process command line arguments and control file.</li> <li>2. Read the package file.</li> <li>3. Process all initialization files, including post-sort initialization files.</li> </ol>	
2. Initialize	<ol style="list-style-type: none"> <li>1. Reset system variables.</li> <li>2. Compute formulas.</li> <li>3. Write queue report files with IO time of <b>After initialization files read</b>.</li> </ol>	

#### Post-sort engine processing in a two-pass engine process, continued

Step	Engine action	Variables populated
For each sort index entry:		
3. Queue	<ol style="list-style-type: none"> <li>1. Read and verify the current sort index entry.</li> <li>2. Find the corresponding sort data file and read back into memory (restoring the state of the engine as it was when the sort data entry was written out).</li> <li>3. Last pass compose—process all remaining late compose objects.</li> <li>4. Pages in documents sorted if required.</li> </ol>	Post-sort only: SYS_BundleCustomer1-5 SYS_BundleCustomerTotal1-5 SYS_BundleInBreak1-5 SYS_BundleInQueue1-5 SYS_BundlePage1-5 SYS_BundlePageTotal1-5 SYS_BundleSheet1-5 SYS_BundleSheetTotal1-5
4. Page IO	<ol style="list-style-type: none"> <li>1. Compute formulas.</li> <li>2. Write report files with IO time of <b>When each page is selected.</b></li> <li>3. Produce pages.</li> </ol>	

## 5.2.3 Engine Processing with Schema Model Data Files

When your application uses a schema model data file that contains XML nodes, use the following engine processing information to help you understand the engine processing order and engine timing.

For more information about customer driver files with XML nodes (such as schema model data files), see *Using Data to Drive an Application* in the Exstream Design and Production documentation.

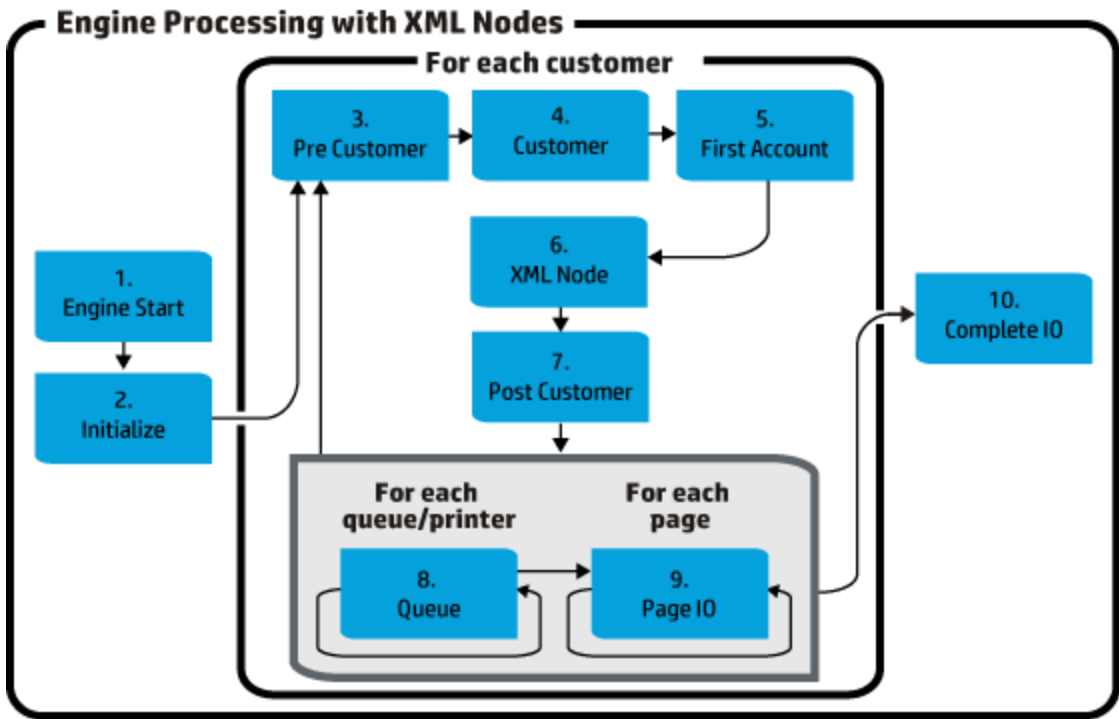
When you review engine processing for your application, keep in mind the following considerations:

- The engine actions are always completed in the same order, regardless of how you design an application. The differences in your output are determined by the way in which you build your application.
- Exstream Live applications require special timing considerations due to differences in variable substitution and rule execution timing. For Exstream Live applications, you can also use special Exstream Live-specific system variables. These system variables have unique behaviors that you should be aware of before you use them.

For more information about special timing considerations in Exstream Live and Exstream Live-specific system variables, see *Designing for LiveEditor* in the Exstream Design and Production documentation.

The following flowchart illustrates an engine processing flow with data files that contain XML nodes:

Engine processing flow with XML nodes



The following table explains in detail what happens during each step in the previous flowchart, and when system variables are populated:

Engine processing steps and engine actions in an engine process with XML nodes

Step	Engine action	System variables populated
1. Engine Start	<ol style="list-style-type: none"><li>1. Process command line arguments and control file.</li><li>2. Read the package file.</li><li>3. Process all initialization files.</li></ol>	



## Engine processing steps and engine actions in an engine process with XML nodes, continued

Step	Engine action	System variables populated
2. Initialize	<ol style="list-style-type: none"> <li>1. Reset system variables.</li> <li>2. Compute formulas.</li> <li>3. Write report files with <b>IO time of After initialization files read.</b></li> </ol>	SYS_DateCurrent SYS_VirtMsg_Contents SYS_VirtMsg_Identifier SYS_VirtMsg_Name SYS_VirtMsg_Priority
Begin customer processing: The engine completes step 3 through step 9 for each customer		
3. Pre Customer	<ol style="list-style-type: none"> <li>1. Set variables from customer records.</li> <li>2. Prepare campaign and message list by firing campaign and message rules (<b>After initial customer data</b>).</li> <li>3. Write report files with IO time of <b>After initial customer data</b>.</li> </ol>	SYS_CustomerBegByte SYS_CustomerBegRecord SYS_DriverFileOffsetBytes SYS_DriverFileOffsetBytesEnd SYS_DriverFileOffsetRecs SYS_DriverFileOffsetRecsEnd SYS_WeightStartGrams SYS_WeightStartOunces
4. Customer	<ol style="list-style-type: none"> <li>1. Set locale and languages.</li> <li>2. Read reference files by IO time.</li> <li>3. Compute formulas.</li> <li>4. Prepare document list firing document rules.</li> <li>5. Prepare page list firing page rules.</li> <li>6. Compose documents.</li> </ol>	SYS_CustomerEffectiveDate SYS_CustomerInRun SYS_CustomerJurisdiction SYS_DateAsOf (user defined) SYS_DDAAOutputUserData (user defined) SYS_DocumentNames SYS_LanguageCustomer SYS_LanguageDefault SYS_LocaleCustomer SYS_LocaleDefault SYS_MktgPagesAllowed SYS_PageInDocument SYS_SubDocInDocument SYS_WeightGrams SYS_WeightOunces
5. First Account		SYS_TableRow

### Engine processing steps and engine actions in an engine process with XML nodes, continued

Step	Engine action	System variables populated
6. XML node	<ol style="list-style-type: none"> <li>1. Read reference files by IO time.</li> <li>2. Set variables from XML node.</li> <li>3. Read reference files by IO time.</li> <li>4. Prepare document list for XML node-based documents.</li> <li>5. Prepare page list firing page rules.</li> <li>6. Set variables until the next new customer is encountered.</li> </ol>	
7. Post Customer	<ol style="list-style-type: none"> <li>1. Prepare campaign and message list by firing campaign and message rules (<b>After all customer data</b>).</li> <li>2. Compute formulas.</li> <li>3. Prepare document list for campaigns firing document rules.</li> <li>4. Prepare page list firing rules.</li> <li>5. Write report files with IO time <b>At end of customer, before campaigns</b>.</li> <li>6. Order and weigh documents.</li> <li>7. Add static back pages and business (ordered) messages.</li> </ol>	<p>SYS_CampaignFirstPage</p> <p>SYS_CampaignPriority</p> <p>SYS_CampaignsQualified</p> <p>SYS_CampaignsSent</p> <p>SYS_SheetUsedByPaperTypes</p> <p>SYS_SheetTotalInDocument</p> <p>SYS_SheetTotalInRun (increments for each customer)</p> <p>SYS_TableRowOnPage</p> <p>SYS_TableRowPage</p> <p>SYS_TableRowTotal</p>
Begin queue processing: The engine completes the following engine actions for each queue that is qualified by a rule		
	<ol style="list-style-type: none"> <li>1. Add footnotes to current pages in document.</li> <li>2. Add marketing messages to current pages in documents.</li> <li>3. Add campaign-driven pages.</li> <li>4. Add any remaining inserts.</li> <li>5. Add blank back pages.</li> <li>6. Count pages and set page numbers.</li> <li>7. Process indexes and tables of content.</li> </ol>	

Engine processing steps and engine actions in an engine process with XML nodes, continued

Step	Engine action	System variables populated
Begin page and composition processing: The engine completes the following engine actions for all pages		
	<ol style="list-style-type: none"><li>1. Order and weigh again.</li><li>2. Add static back pages and business (ordered) messages.</li><li>3. Count pages and set page numbers.</li></ol>	
Continue queue processing: The engine completes step 10 through step 11 for each queue		

## Engine processing steps and engine actions in an engine process with XML nodes, continued

Step	Engine action	System variables populated
10. Queue	<ol style="list-style-type: none"> <li>1. Set queue system variables.</li> <li>2. Write report files with IO time of <b>Queue break and end of queue</b>.</li> <li>3. Run queue rules and perform final document selection per queue.</li> <li>4. Process all objects flagged as late compose (Last pass compose).</li> <li>5. Evaluate recipient data and update content as necessary for recipient copies.</li> <li>6. Sort pages in documents if required.</li> </ol> <p>While many variables can be used on the page with late object timing, step 10, or Queue, triggers their use (update or late composition)</p>	<div> <div>SYS_BreaksInQueue</div> <div>SYS_PagePrintedValue</div> </div> <div> <div>SYS_ByteInBreak</div> <div>SYS_PageStart</div> </div> <div> <div>SYS_ByteInQueue</div> <div>SYS_PageTotalInDocument</div> </div> <div> <div>SYS_CurrentProfile</div> <div>SYS_PageTotalInRun (increments for each customer),</div> </div> <div> <div>SYS_CustomerChecksum</div> <div>SYS_PageTotalPhysicalInDocument</div> </div> <div> <div>SYS_DocPrintedValue</div> <div>SYS_PageTotalPrinted</div> </div> <div> <div>SYS_DocumentChecksums</div> <div>SYS_PaperTypes (increments for each customer)</div> </div> <div> <div>SYS_DocumentInBreak</div> <div>SYS_PDLInQueue</div> </div> <div> <div>SYS_DocumentInQueue</div> <div>SYS_PrinterInQueue</div> </div> <div> <div>SYS_DocumentTotalInRun (increments for each customer)</div> <div>SYS_QueueCurrent</div> </div> <div> <div>SYS_InserterBinPriorities</div> <div>SYS_QueueFileName</div> </div> <div> <div>SYS_InserterBins</div> <div>SYS_QueueToCustomer</div> </div> <div> <div>SYS_InserterBinString</div> <div>SYS_RecordInBreak</div> </div> <div> <div>SYS_InserterInQueue</div> <div>SYS_RecordInQueue</div> </div> <div> <div>SYS_MktgPagesAdded</div> <div>SYS_SectionName</div> </div> <div> <div>SYS_MsgContents</div> <div>SYS_SheetInBreak</div> </div> <div> <div>SYS_MsgIdentifier</div> <div>SYS_SheetInDocument</div> </div> <div> <div>SYS_MsgName</div> <div>SYS_SheetInQueue</div> </div> <div> <div>SYS_NumBannersInQueue</div> <div>SYS_SortIndex</div> </div> <div> <div>SYS_PageFlows</div> <div>SYS_SubDocumentInBreak</div> </div> <div> <div>SYS_PageInBreak</div> <div>SYS_SubDocumentInQueue</div> </div> <div> <div>SYS_PageInQueue</div> <div>SYS_TotalSubDocsInDocument</div> </div> <div> <div>SYS_PagePhysicalInDocument</div> <div>SYS_VolSerInQueue</div> </div> <p>Variables that may be modified by other processes at later events:</p> <div> <div>SYS_DocumentNames</div> <div>SYS_MktgPagesAllowed</div> <div>SYS_PageInDocument</div> <div>SYS_SubDocInDocument</div> </div>

Engine processing steps and engine actions in an engine process with XML nodes, continued

Step	Engine action	System variables populated
		SYS_WeightGrams SYS_WeightOunces
Continue page and composition processing: The engine completes the following step for each page		
11. Page IO	<ol style="list-style-type: none"><li>1. Compute formulas.</li><li>2. Write report files with IO time of <b>When each page is selected</b>.</li><li>3. Output pages.</li></ol>	
12. Complete IO	Write report files with IO time of <b>Completion of all customers</b> .	

## 5.2.4 Post-Sort Engine Processing with Schema Model Data Files

**Note:** You must have licensed the Output Sorting and Bundling module to execute two-pass engine processing.

For information on the Output Sorting and Bundling module, see *Creating Output* in the Exstream Design and Production documentation.

In two-pass engine processing (separately known as pre-sort engine processing and post-sort engine processing), the engine is run twice in order to sort and bundle your output for postal discounts. During the first engine run (pre-sort engine processing), the engine produces customer data for the run that you can manipulate with third-party software to sort, cleanse, and update. After user sorting, you run the engine a second time (post-sort engine processing) in order to produce output according to the sorted information.

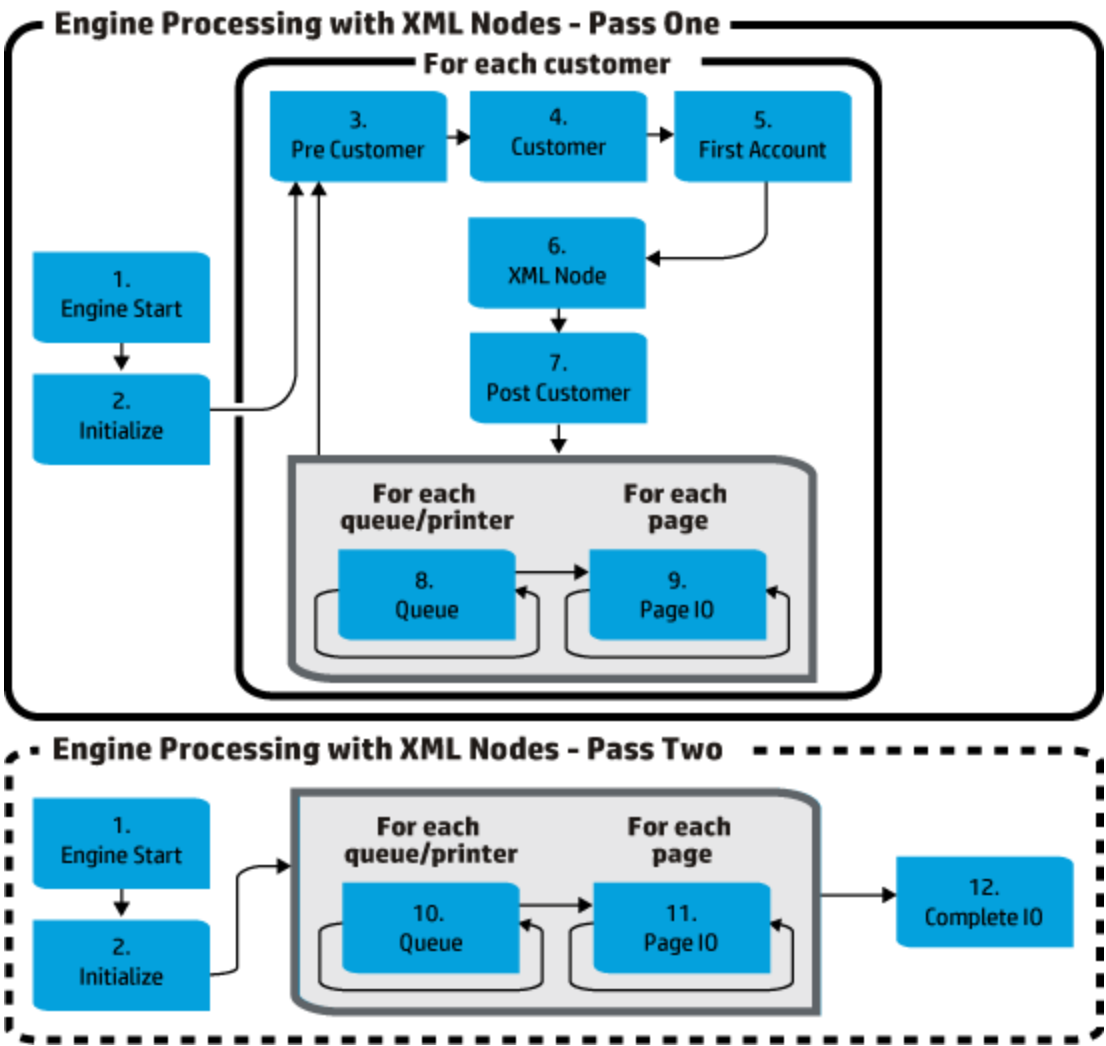
With two-pass engine processing, keep in mind the following considerations:

- All documents and pages are stored in pre-sort engine processing and re-read in post-sort engine processing.
- Only late compose objects are drawn in post-sort engine processing.
- Post-sort engine processing does not read the driver or reference data files, nor does it perform many of the events of pre-sort engine processing.
- Sort data file entries are a capture of the engine before the final late compose step is performed. Each entry contains the following:

- The drawn documents and pages
  - The design objects to be drawn in post-sort engine processing
  - Any variables that are used in post-sort engine processing
- 
- When each customer's sort data entry is read, all post-sort engine processing variables are reset to their pre-sort engine processing value. Simple tasks like user-defined counters will seem to work incorrectly. Thus, variables that are used in post-sort engine processing, but that are not to be restored to their pre-sort engine processing values for each entry must be flagged as such. There are various options for doing this, such as using a post-sort initialization file.

The following flowchart illustrates how engine processing differs when you use the two-pass engine process that is required for output sorting and bundling:

Two-pass engine processing flow with XML nodes



Pre-sort engine processing in a two-pass engine process

Step	Engine action
1-7	<p>In pre-sort engine processing for two-pass engine processes, the engine actions and system variables that are populated for step 1 through step 7 are identical to engine processing with XML nodes.</p> <p>For more information about engine processing for general data and data sections, see <a href="#">"Engine Processing with Schema Model Data Files"</a> on page 103.</p>

#### Pre-sort engine processing in a two-pass engine process, continued

Step	Engine action
8. Queue	<ol style="list-style-type: none"> <li>1. Set the queue system variables.</li> <li>2. Write report files with IO time of queue break and end of queue.</li> <li>3. Run queue rules and perform final document selection per queue.</li> <li>4. Last pass compose—process all objects flagged as late compose. Note that objects flagged as after document sorting or that use variables affected by document sorting (for example SYS_PageInDocument) are not composed yet.</li> <li>5. Evaluate recipient data and update content as necessary for recipient copies.</li> <li>6. Write customer and recipient entries to the sort index file.</li> </ol>
After all customers:	
10. Complete IO	Write report files with IO time of <b>Completion of all customers</b> .

At the end of pre-sort processing, you can sort the sort index file and change the fields that can be updated.

During post-sort engine processing, the engine produces output based on the information that was sorted in pre-sort engine processing.

The following table explains in detail what happens during each step in of post-sort engine processing, as shown in the previous flowchart, and when system variables are populated:

#### Post-sort engine processing in a two-pass engine process

Step	Engine action	Variables populated
1. Engine Start	<ol style="list-style-type: none"> <li>1. Process command line arguments and control file.</li> <li>2. Read the package file.</li> <li>3. Process all initialization files, including post-sort initialization files.</li> </ol>	
2. Initialize	<ol style="list-style-type: none"> <li>1. Reset system variables.</li> <li>2. Compute formulas.</li> <li>3. Write queue report files with IO time of <b>After initialization files read</b>.</li> </ol>	



#### Post-sort engine processing in a two-pass engine process, continued

Step	Engine action	Variables populated
For each sort index entry:		
3. Queue	<ol style="list-style-type: none"> <li>1. Read and verify the current sort index entry.</li> <li>2. Find the corresponding sort data file and read back into memory (restoring the state of the engine as it was when the sort data entry was written out).</li> <li>3. Last pass compose—process all remaining late compose objects.</li> <li>4. Pages in documents sorted if required.</li> </ol>	Post-sort only: SYS_BundleCustomer1-5 SYS_BundleCustomerTotal1-5 SYS_BundleInBreak1-5 SYS_BundleInQueue1-5 SYS_BundlePage1-5 SYS_BundlePageTotal1-5 SYS_BundleSheet1-5 SYS_BundleSheetTotal1-5
4. Page IO	<ol style="list-style-type: none"> <li>1. Compute formulas.</li> <li>2. Write report files with IO time of <b>When each page is selected.</b></li> <li>3. Produce pages.</li> </ol>	

## 5.3 Managing Production Engine Processing Memory

Depending on the type of documents your company creates, you must consider how much memory is available on your platform and how available memory can affect engine processing. For example, you can encounter memory issues during production when processing transaction-intensive documents, such as large bank statements or financial portfolios. You can eliminate many memory and performance issues by installing the 64-bit version of the production engine.

Exstream offers the following production engines on all platform types:

- 32-bit—You can use this production engine version with 32-bit capable hardware on a 32-bit or 64-bit operating system.
- 64-bit—You can use this production engine version with 64-bit capable hardware on a 64-bit operating system.

For more information about supported platforms for 32-bit and 64-bit production engines, see *Installation and Upgrade Information* in the Exstream Design and Production documentation.

The amount of memory available for processing varies based on your production platform and server setup.

If you are producing applications using the 32-bit Exstream engine, use the examples in the following table as a guideline to determine the maximum memory usage your operating system can support.

32-bit Exstream production engine  
maximum memory support

Operating system	Memory available
Windows 64-bit	4GB
Windows 32-bit	2GB
Sun	3GB
Linux 64-bit	4GB

**Caution:** The table is a reference tool for troubleshooting memory management issues only. It is not a guarantee of performance on all platforms and all configurations.

To manage memory and avoid memory allocation issues when producing applications on the 32-bit version of the Exstream production engine, use the following engine switches:

- MEMORYSAVE
- MEMORYCACHE
- CACHETABLE

If you run the 64-bit version of the Exstream production engine, the MEMORYSAVE, MEMORYCACHE, and CACHETABLE engine switches are not necessary to manage memory.

To use these three engine switches to better manage memory usage on the 32-bit version of the Exstream production engine, complete the following tasks as needed:

- [“Resetting Memory with the Start of Each New Section/Customer” below](#)
- [“Writing Table Row Data to Specified Temporary Files” on the next page](#)
- [“Applying Memory Management Techniques to Maximize Engine Potential for the Production Engine on z/OS” on page 116](#)

### 5.3.1 Resetting Memory with the Start of Each New Section/Customer

You can use the MEMORYSAVE engine switch to reset memory at the start of each new section or customer.

Without this switch, the engine uses memory to store variables and composed pages at the largest variable. For example, if one customer contains 600 pages in a document, the engine keeps the allocation for page memory at this level, even if the next customer only requires a fraction of that amount.

With the MEMORYSAVE switch, the engine releases the memory allocated to variables with each new section. This dynamic approach to memory management can reduce the processing speed for the customer with the large memory requirement but, overall, it eases memory requirements in most large runs, especially those with a large number of transactions.

When you use the MEMORYSAVE switch on the command line or in a control file, the engine waits to begin the reallocation of memory until the run hits the threshold you set as a numeric value. This number represents a transaction count (an element count in a single array variable). Typically, the value is set to 20,000. This value refers to the number of records read before writing to the file. This value can be changed, depending on the application.

For more information about the MEMORYSAVE switch, see *Switch Reference* in the Exstream Design and Production documentation.

## 5.3.2 Writing Table Row Data to Specified Temporary Files

If you have large transaction-based tables, use the MEMORYCACHE switch to save memory by writing table row data to a temporary file you specify.

**Tip:** For optimum memory usage, you can use both MEMORYCACHE and MEMORYSAVE switches.

The engine writes all subsequent row data to a file instead of memory when the row count for any one table exceeds one of the following conditions:

- 250,000 (the CACHETABLE default and the value used if you do not use the CACHETABLE switch)
- The number you specify with the CACHETABLE switch

When the engine reaches the end of the table, it sends the stored row data into the print stream along with pagination, header/footer, and other page design information. The engine then resets the file so it is ready for the next large table in the run. The engine uses additional files if it reaches the memory limit or if it encounters read/write errors.

For more information about the MEMORYCACHE and CACHETABLE switches, see *Switch Reference* in the Exstream Design and Production documentation.

### 5.3.3 Applying Memory Management Techniques to Maximize Engine Potential for the Production Engine on z/OS

To improve production engine performance on z/OS, you can apply some memory management techniques. The z/OS platform has several specific techniques that can help you maximize the potential of the engine on the z/OS platform. You can use the following as techniques to improve performance:

- Applying z/OS-specific configurations to improve engine performance, including the following:
  - Using C/C++ memory calls
  - Use the IBM language environment to manage stacks and heaps
  - Fine-tune an application using Language Environment (LE) tuning parameters
- Specifying the MEMORYCACHE engine switch
- Specifying estimated z/OS file size for defined queues
- Pre-loading reference files

### Applying z/OS-Specific Configurations for Engine Optimization

You can apply specific configurations on the z/OS platform to optimize engine performance in the following ways:

- Use C/C++ memory calls in the production environment.
- Use the IBM Language Environment (LE) to manage stacks and heaps in z/OS.
- Use LE tuning parameters to fine-tune your application.

### Using C/C++ Memory Calls in the Production Environment

You must use C/C++ memory call in z/OS to manage the allocation of memory. If your company requires you to produce extremely large and complex applications, the production environment allocates a substantial amount of the memory available on your platform. In this situation, you must utilize memory calls and enable the MEMORYSAVE engine switch to inform the engine to free the memory you need to avoid potential processing errors.

For more information about the MEMORYSAVE switch, see *Switch Reference* in the Exstream Design and Production documentation.

Use the following C/C++ standard memory calls when operating in the production environment on z/OS:

- `new()`
- `delete()`
- `malloc()`
- `realloc()`
- `free()`

To improve memory usage in large production runs on the 32-bit z/OS engine, it is best to use the MEMORYSAVE switch in your control file. The MEMORYSAVE switch lets you control when the engine makes memory available.

## Using the IBM Language Environment to Manage Stacks and Heaps

Since the production environment uses standard memory calls, you can use the IBM LE to manage its stack and heaps. LE provides run-time options to control certain aspects of processing.

Exstream has no "memory looping;" it requests memory from LE. If you overload memory, problems can occur in how LE uses extra CPU cycles to acquire it.

You can set LE to report how it uses memory and to configure your JCL for better performance. Different package files use memory differently, so before you move an application to production, you can run the LE reports to configure memory usage optimally.

To generate a storage report and run-time options report for program ENGEXE, specify the following:

```
//G01 EXEC PGM=ENGEXE,PARM='RPTSTG(ON),RPTOPTS(ON)/'
```

You can apply LE run-time options in the PARM argument step in your JCL. Anything before a forward slash is sent to LE as an option; anything after the slash is sent as an argument.

```
//[stepname] EXEC PGM=ENGEXE,  
//PARM='[run-time options]/[program parameters]'
```

For LE report memory configuration and usage reports (including tuning recommendations) use the following:

```
//[stepname] EXEC PGM=ENGEXE,  
//PARM='RPTSTG(ON),RPTOPTS(ON)/-USECONTROL=YES'
```

RPTSTG(ON) shows the storage the application uses at run time. Use this report to help decide what values and suboptions to indicate at run time.

**Note:** Because it increases run time, you typically only use RPTSTG(ON) to assist with application development.

RPTOPTS(ON) shows the run-time options in effect during application run time.

## Using LE Tuning Parameters to Fine-Tune an Application

You can use the following LE tuning parameters to fine-tune your application in the production environment:

- ANYHEAP
- HEAP
- STACK. HEAP
- HEAPPOLS

The main LE tuning parameters are ANYHEAP, HEAP, HEAPPOLS, and STACK. HEAP and HEAPPOLS are key when tuning an application to improve memory usage and performance.

The ANYHEAP option controls the allocation of library heap storage not restricted to below the 16M line. This option is always in effect. The IBM default is as follows:

```
ANYHEAP(16K, 8K, ANYWHERE, FREE)
```

The following is another example:

```
ANYHEAP(640K, 32K, ANY, FREE)
```

The HEAP option controls the allocation of initial and additional heaps and details how that storage is managed. This option is always in effect. The default is as follows:

```
HEAP(32K, 32K, ANYWHERE, KEEP, 8K, 4K)
```

For smaller applications, the LE reports require setting the heap to the following:

```
//PARM= 'HEAP(5M, 256K, ANY, FREE, 8K, 4K) / -USECONTROL=YES '
```

For larger applications, set the heap to the following:

```
//PARM= 'HEAP(15M, 512K, ANY, FREE, 8K, 4K) / -USECONTROL=YES '
```

The following table shows the heap suboptions.

Suboptions for the Heap tuning parameter

Value	Suboption
init_size	This suboption establishes the minimum initial allocation of heap storage. Designate as n, nK, or nM bytes of storage. The actual amount of allocated storage rounds up to the nearest multiple of 8-bytes.
incr_size	This suboption establishes the minimum size of any subsequent increment to the heap storage. Designate as n, nK, or nM bytes of storage. The actual amount of allocated storage rounds up to the nearest multiple of 8-bytes.

### Suboptions for the Heap tuning parameter, continued

Value	Suboption
ANYWHERE/ANY	This suboption allocates heap storage anywhere in storage. On systems that support bimodal addressing, you can allocate storage above or below the 16M line. If no storage is available above the line, storage is obtained below the line. On systems that do not support bimodal addressing, LE ignores this option and allocates the heap storage below 16M. The abbreviation for ANYWHERE is ANY.
BELOW	This suboption allocates heap storage below the 16M line in storage that is accessible to 24-bit addressing. The BELOW suboption causes the HEAPPOLS option to be ignored because HEAPPOLS can only run above the line.
KEEP	This suboption stipulates that storage allocated to HEAP increments is not released when the last of the storage is freed.
FREE	This suboption stipulates that storage allocated to HEAP increments is released when the last of the storage is freed. The FREE suboption has no impact on heap increments containing a HEAPPOLS cell pool.
initsz24	This suboption establishes the minimum initial size of the heap storage obtained below the 16M line for applications running with ALL31(OFF) when these applications specify ANYWHERE in the HEAP run-time option. Designate as n, nK, or nM number of bytes. The amount of storage rounds up to the nearest multiple of 8-bytes. initsz24 applies to the initial heap and other heaps created with the CEECRHP call not allocated strictly below the 16M line.
incrsz24	This suboption establishes the minimum size for any subsequent increment to the heap area obtained below the 16M line for applications running with ALL31(OFF) when these applications specify ANYWHERE in the HEAP run-time option. Designate as n, nK, or nM number of bytes. The amount of storage rounds up to the nearest multiple of 8- bytes. incrsz24 applies to the initial heap and other heaps created with the CEECRHP call not allocated strictly below the 16M line.

For C languages, LE has a faster heap manager called HEAPPOLS. This causes faster runs but use slightly more memory.

HEAPPOLS(ON,8,10,32,10,128,10,256,10,1024,10,2048,10)

The HEAPPOLS option controls an optional heap storage management algorithm (heap pools) and is used to improve performance of multi-threaded C/C++ applications with high use of standard memory calls.

The following table describes the suboptions available for heap pools.

### Suboptions for the Headpools tuning parameter

Value	Suboption
OFF	This suboption establishes that LE does not use the heap pool manager.
ON	This suboption establishes that LE uses the heap pool manager to manage heap storage requests against the initial heap.
[ cell size]	This suboption indicates the size of cells in a heap pool. The cell size must be a multiple of 8, up to 2048. Cell sizes 1K and 2K are also allowed.
[ percentage]	This suboption percentage of HEAP run-time option's init_size value to be used as the size for the heap pool and any extents. The percentage must be between 1 and 90.

Stack controls the allocation of the thread's stack storage. The default is as follows:

STACK(128K,128K,BELOW,KEEP)

For more information about tuning LE, see Language Environment for OS/390 & VM - Programming Guide SC28-1939 and Programming Reference SC28-1940, available from IBM.

## Specifying the MEMORYCACHE Engine Switch as an ESDS VSAM File

In a z/OS production run, the file you specify with the MEMORYCACHE switch must be an ESDS VSAM file. The size of the file must accommodate row data for your largest customer. The VSAM cluster must be defined as reusable prior to use (default is non-reusable).

**Note:** With extremely large tables, you can use multiple MEMORYCACHE entries with different file names. The engine writes table row data to the next file in the list when the first file fills up.

The following sections of sample code are necessary to run an application using MEMORYCACHE. These are presented in the order they must appear in the code.

MEMORYCACHE code samples for z/OS

Description	Location	Sample
Define cluster	Before the run engine command	//Define EXEC PGM=IDCAMS//SYSPRINT DD SYSOUTPUT=**//Define VSAM KSDS DatasetDEFINE CLUSTER(NAME(SVCS.TEST.KSDS) -RECORDSIZE(32 32000) -CYLINDERS (100 100) -REUSE -VOLUMES (XBIG01) -NONINDEXED)/*
Reference the JCL	In the run engine command	//CACHING DD DSN=SVCS.TEST.KSDS,DISP=SHR
Control file	Referenced by the sample JCL above	-PACKAGEFILE=DD:EXPACKAG-MESSAGEFILE=DD:EXMSGGS - REPORTFILE=DD:EXREPORT -RUNMODE=PRODUCTION -TRACKIN=DISABLE -TRACKOUT=FILE -REPORT=CUSTOMER - MEMORYCACHE='SVCS.TEST.KSDS' *MEMORYCACHE=DD:CACHING -CACHETABLE=3000 -MEMORYSAVE=5000-PACKAGEFILE=DD:EXPACKAG- MESSAGEFILE=DD:EXMSGGS -REPORTFILE=DD:EXREPORT -RUNMODE=PRODUCTION -TRACKIN=DISABLE -TRACKOUT=FILE - REPORT=CUSTOMER -MEMORYCACHE=-SVCS.TEST.KSDS- *MEMORYCACHE=DD:CACHING -CACHETABLE=3000 -MEMORYSAVE=5000

## Specifying Estimated z/OS File Size for Defined Queues

If you use a z/OS platform for production, you must specify the output file size. Since z/OS allocates in tracks and cylinders, allocating too large a file size can waste space. To use Direct Access Storage Device (DASD) hard drive space efficiently, select a conservative size. However, choosing too small a size can result in slower production speeds. The default, 200K, is a moderate file size.



To specify the size of an output file:

1. Open an output queue in the Property Panel.
2. Specify the expected size for output files in kilobytes in the **MVS file size (KB)** box. If you do not specify a value, the size of each file is set at 200KB.

#### z/OS file size on queue properties

The screenshot shows a dialog box with two main sections: 'Description' and 'Rule'. The 'Description' section on the left contains fields for 'PDF output queue', 'Normal full production' (selected in a dropdown), 'PDF' (selected in an 'Output' dropdown), a checked 'Do not create print file' checkbox, 'No Variable' (selected in a 'Record prefix' dropdown), and 'None' (selected in a 'Jogging' dropdown). The 'Rule' section on the right contains a large empty text box, 'No Variable' (selected in a 'Variable for file naming' dropdown), 'C:\Users\Public\Documents\OpenText\Exstream\Output Files' (selected in a 'Test file' dropdown), 'PDFOUT' (selected in a 'Production file' dropdown), 'No Connector' (selected in both 'Test output connector' and 'Production output connector' dropdowns), and a circled 'MVS file size (KB)' dropdown with '200' selected.

### Using an Algorithm in the Production Environment to Set the z/OS File Size

The production environment uses the following algorithm to convert the specified value to TRKS and CYLS.

- If the z/OS file size is less than 1200KB, use TRACKS and the primary amount is equal to the amount / 50.
- Otherwise, use CYLINDERS and the primary amount is equal to the amount / 800.
- Secondary storage is set to the primary / 3.
- Disposition is (CATLG, CATLG).
- Excess space is released (RLSE).
- A model dataset is opened and closed to get the LRECL, BLKSIZE, RECFM, and VOLSER.

**Note:** The model dataset is specified by the production file. It is only used to get LRECL, BLKSIZE, RECFM, and VOLSER to be used when allocating data sets dynamically.

## Preloading Reference Files on z/OS

Use the LOADREFFILE engine switch to preload reference files into VSAM in z/OS production. Using the LOADREFFILE switch can reduce the CPU time needed to run applications on the mainframe. The LOADREFFILE=filename switch on the z/OS engine places the engine in a run mode that only loads the lookup files and then ends. The customer data files are not read and no output is created. The file name specified is the name of the reference file to load. Use LOADREFFILE=ALL to preload all of the files.

**Note:** You can load VSAM files once and run multiple times.

# Chapter 6: Running the Production Engine

The production engine is required to run Exstream in the production environment. You run the engine from the command prompt, through scripts or Job Control Language (JCL). You must use engine switches and specify them either on the command line when you execute the program or in a control file to run the engine. In most business environments, the production engine is run on a separate platform from the design environment. Logistics, workflow, and other factors determine this setup.

You can run the production engine on any of the following platforms:

- Windows
- UNIX
- Linux
- AS/400
- z/OS

For more information about detailed platform specifications, see *Installation and Upgrade Information* in the Exstream Design and Production documentation.

**Note:** If you run the engine for the production environment from a design workstation, the engine is limited to 120 pages per minute. If you run the design engine, it runs at full speed.

## 6.1 Preparing the Application for Production Engine Processing

After you complete all design and testing phases of your application, you must prepare the application for production engine processing. This section discusses the following topics:

- [“Retaining Naming Conventions for Directory and Production File Names” on the next page](#)
- [“Transferring Package Files from the Design Environment to the Production Environment” on the next page](#)

## 6.1.1 Retaining Naming Conventions for Directory and Production File Names

When you are working from the command prompt, it is important to retain the naming conventions for the platform you are using. Specify the directory and production file names or arguments according to the information in the table.

Directory and file specification naming conventions

Platform	Naming convention
Windows	<p>C:\dir1\...\dirn\filename</p> <p>The drive specification, C:, is optional. If you omit the drive specification, the current working drive is assumed. Each directory specification and file name can be up to 128 characters in length.</p> <p>If you begin with a backslash (\), the file is relative to the root directory. If you do not begin the file specification with a backslash, the file specification is relative to your current working directory.</p>
UNIX/Linux	<p>/dir1/... /dirn/filename</p> <p>If you do not begin the file specification with a slash (/), the file is relative to your home directory. If you begin with a slash, the file is relative to the root directory. UNIX directory and file names are case-sensitive. Make sure you verify all directories and file names for spelling and case before using them. If the names do not match, you receive an error message.</p>
AS/400	<p>On AS/400, make sure that all files used in the application exist in the same directory.</p> <p>On the <b>Production Data Source</b> tab, use local names when defining file names. Directory and file names are case-sensitive.</p>
z/OS	<p>Directory and file names on z/OS must be in all capital letters. For example, you use names like DD : DATAOUT or ' HLQ . XXXX . XXX ' on z/OS.</p> <p>Use the following naming conventions when working with z/OS files:</p> <ul style="list-style-type: none"> <li>z/OS datasets or files must be comprised of up to eight segments of up to eight characters separated by a period. The first segment is called the High Level Qualifier (HLQ). The HLQ defaults to your login name and is automatically prepended to any data set name (DSN) not enclosed in single quotes. For example if you are logged in as P390A, then TESTING . CONTROL is the same as ' P390A . TESTING . CONTROL ' .</li> </ul> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p><b>Note:</b> Segments do not represent a directory structure.</p> </div> <ul style="list-style-type: none"> <li>Characters must be A-Z, 0-9, and \$.</li> <li>The directory structure must be flat and entered into the system catalog.</li> </ul>

## 6.1.2 Transferring Package Files from the Design Environment to the Production Environment

Since, most organizations use a different platform to operate the production environment than the design environment, you must transfer package files from the design environment to the platform operating the production environment.

Before you transfer package files, make sure you have the following:

- The package file
- All data files used in the application
- Any control files or script files to be used by the application

To transfer package files to Windows, UNIX, Linux, and z/OS systems, open your FTP program in `bin` mode and transfer the package file.

If you transfer a package file to a Windows system, you can use standard Windows copy functionality as an optional method to transfer the package file.

To transfer package files to an AS/400, before uploading to the AIX engine directory, remove all carriage returns (line feeds) from text files. To do this, replace all `X'0D0A'` commands with `X'0A'` commands.

If you want to upload files that can be read by a text editor, use the FTP command `ascii`.

For files that cannot be read by a text editor, complete the following steps:

1. Open your FTP program in `bin` mode.
2. Enter the FTP command `quote site namefmt 1`.
3. Use FTP to access the AS/400 PASE environment.
4. Change the directory to the one where the AIX engine is located (for example: `cd /Exstream/Tar`).
5. Upload the package file to the directory where the AIX engine is located.
6. Upload any necessary TXT or DAT files to the same directory.
7. Make sure the carriage returns have been removed from these files.
8. Upload the engine control file to the same directory.
9. If you want to run the engine using a script file, upload the script file at this time.

## 6.2 Using the Command Prompt to Execute a Production Engine Run

When your application reaches the production stages, you use the command prompt to execute an engine run. The following must be specified when executing an engine run from your platform's command prompt:

- The `PACKAGEFILE` engine switch—The `PACKAGEFILE` engine switch tells the engine which package file to use to compose the application. You must specify the location of the

package file as the argument for this engine switch. Make sure the features contained within the package file are authorized in the license key. If the engine encounters a feature not authorized by the license key, the engine stops processing the application and you receive an error message.

- The `MsgResource_<language>.dat` file—The engine uses the message resource file to communicate messages to the user. If the engine cannot find the message resource file, you receive an error message and the engine stops processing.

## 6.2.1 Executing an Engine Run

To execute an engine run from the Windows command prompt:

1. Click **Start > Program Files > Accessories > Command Prompt**, or click **Start > Run**, and enter `cmd` as the argument.
2. Change to the Exstream directory and enter the following command:  
`prodengine -PACKAGEFILE`
3. Add any additional engine switches necessary to run your application through the engine. Make sure you enter a space between each engine switch.
4. Press `ENTER` to run the engine.

The customer numbers in each application appear in the window as they are processed. When the engine completes processing, you return to the command prompt.

## 6.2.2 Executing an Engine Run From a UNIX/Linux Shell Prompt or Shell Script

Before you run the engine, access the UNIX/Linux server using a Telnet connection. You can run the engine using one of the following:

- Shell prompt
- Shell script

To execute an engine run using a shell prompt:

1. At the prompt, enter `cd` followed by a space.
2. After the space, specify the directory where the engine is located.
3. Press `ENTER` to change the directory.

The change is successful when you are returned to the command prompt.

4. Enter `Engine` after the prompt, followed by a space.

5. After the space, add the mandatory PACKAGEFILE switch.

In the example the entire path is added to the specified file name:

```
-PACKAGEFILE=/Exstream/INPUT/package.pub
```

6. Enter a space after the -PACKAGEFILE switch.
7. Add any additional switches necessary to run the engine.

Remember to enter a space between each switch.

8. Press `ENTER` to execute the engine.

The customer numbers appear in the window as they are processed. When the engine finishes processing, the prompt returns.

You can also run the engine using a shell script, which contains all of the necessary commands and specific engine switches necessary to execute an engine run. With a shell script, you can execute an engine run using a single command at the shell prompt. The following is a sample shell script with comments.

```
#!/bin/sh
# This script links the file input.dat to the pseudo-file DD:INPUT1
# then links optional.rule.file to the pseudo-file DD:INPUT8
# then executes Exstream (called ENGINE in this example)
# with the command options -controlfile=control.file and -
messagefile=message.summary
# A file DD:OUTPUT1 is created and then renamed to output.file
#
# This example assumes that all files are in the current directory
# of execution and that ENGINE is in the execution path of the
# current user.
# Remember of course these values are case sensitive.
#!/bin/sh
# This script links the file input.dat to the pseudo-file DD:INPUT1
# then links optional.rule.file to the pseudo-file DD:INPUT8
# then executes Exstream (called ENGINE in this example)
# with the command options -controlfile=control.file and -
messagefile=message.summary
# A file DD:OUTPUT1 is created and then renamed to output.file
#
# This example assumes that all files are in the current directory
# of execution and that ENGINE is in the execution path of the
# current user.
# Remember of course these values are case sensitive.

ln input.data DD:INPUT1
ln optional.rule.file DD:INPUT8
ENGINE -controlfile=control.file -messagefile=message.summary
mv DD:OUTPUT1 output.file
```

## 6.2.3 Executing an Engine Run From an AS/400 Shell Environment

Using a control file to run the package file through the engine:

1. Call the interactive shell environment from the AS/400 from a Telnet session.

For example: `call qp2term`

2. Change the directory to the AIX engine location.
3. Call the engine and specify the control file on the command prompt.

For example: `./Engine -CONTROLFILE=controlprod.opt`

4. Press `ENTER` to execute the engine run.

To retrieve your output files:

1. Verify the output file has been created by the engine.

For example: `ls`

2. Using FTP, change to the local directory where you want to place the output file(s).

For example: `lcd C:\FTP_Files\`

3. Access the directory on the AS/400 where the AIX engine is located.

In the example, this directory is `/exstream/tar`.

4. Using the `get` command, copy the output file from the directory on the AS/400 to your local workstation.

For example: `get AS400.pdf`

For more information about output and output files, see *Creating Output* in the Exstream Design and Production documentation.

## 6.2.4 Executing an Engine Run on z/OS Command Line Interface

Like all other platforms, you execute engine runs on the z/OS platform by entering commands on the command line interface. To execute a production engine run on the z/OS platform, you must complete the following tasks:

1. [“Specifying File Names Using z/OS Commands” on the next page](#)
2. [“Configuring JCL Files to Execute a Production Run” on page 130](#)



To enhance engine performance, you can also complete the following optional task as needed:

- [“Configuring a JCL to Load and Run Multiple Package Files as VSAM Package Files” on page 133](#)

## Specifying File Names Using z/OS Commands

To specify z/OS file names using z/OS commands, you can use several formats. For example:

`-COMMAND=DD:ddname`

`-COMMAND=dd:ddname(member)`

`-COMMAND=qualifier.qualifier.qualifier`

`-COMMAND=qualifier.qualifier.qualifier(member)`

File names include the following:

- **ddname**—The data definition of the file in your JCL
- **member**—The member name of a PDS
- **qualifier**—A z/OS data set

The slash (/) character is supported as an argument specifier on the z/OS platform. It is used to define parameters or switches on the command line interface.

## Configuring JCL Files to Execute a Production Run

To execute an engine run on a z/OS system, you must configure a JCL file. Use the sample JCL as a guide to configuring your own JCL file to run the engine. Engine-specific JCL commands begin with line 9 in the example.

```
//VERAFP JOB (999,P0K), ",  
//      TIME=1440,  
//      NOTIFY=-&SYSUID;,  
//      REGION=0M,  
//      CLASS=A,  
//      MSGCLASS=H,  
//      MSGLEVEL=(1,1)  
//*  
//EBCEXE EXEC PGM=ENGEXE,  
//      PARM=' -USECONTROL=YES '  
//STEPLIB DD DSN=P390A.EXSTREAM.LOAD ,DISP=SHR  
//DLMSGRES DD DSN=P390A.DMSGENUS,DISP=SHR  
//EXMSGSG DD DSN=P390A.EXSTREAM.VERMSG(MSGAFP2) ,DISP=SHR  
//BANKDATA DD DSN=P390A.EXSTREAM.VERDATA(BANKDATA) ,DISP=SHR  
//EXPACKAG DD DSN=P390A.EXSTREAM.VERPACK(PACKAFP) ,DISP=SHR  
//EXREPORT DD DSN=P390A.EXSTREAM.VERRPT(RPTAFP2) ,DISP=SHR  
//EXCONTRL DD DSN=P390A.EXSTREAM.VMSCTRL,DISP=SHR  
//EXTRACK DD DSN=P390A.EXSTREAM.VERTRK(TRKAFP2) ,DISP=SHR  
//EXOUTPUT DD DSN=P390A.EXSTREAM.VERAFP(EXAFP2) ,DISP=SHR
```

### Configuring Engine-Specific JCL Commands

The following is an explanation of common JCL commands as they relate to running the engine on the z/OS platform. Each line explains the JCL command and the subsequent action the engine takes. Remember that all file names are examples. When you create and use a JCL file to run a package file through the engine, you must make sure you have accurate file names and locations for your production run. The explanation begins with line 9 of the previous sample JCL:

To configure engine-specific JCL commands, complete the following steps:

1. Load the engine executable using the following command:

```
//EBCEXE EXEC PGM=ENGEXE,
```

In the sample JCL, the load module is the PDS, P390A.EXSTREAM.LOAD, identified by the //STEPLIB argument that follows.

2. Specify if you want to use a control file to define parameters for the load module using the following command:

```
//PARM=' -USECONTROL=<YES or NO> '
```

In the example, the JCL tells the ENGEXE load module to use a control file. Since the name of the control file was not set, the engine will use the default control value of DD:EXCONTRL.

In this example, the DD:EXCONTROL references P390A.EXSTREAM.VMSCTRL

3. Identify the location of the load module using the following command:

```
//STEPLIB DD DSN= <PDS>,DISP=SHR
```

In the sample JCL, P390A.EXSTREAM.LOAD is the location for the load module.

4. Specify the message resource file using the following command:

```
//DLMSGRES DD DSN= <PDS>.DMSGENUS
```

DLMSGRES is the default DD value in z/OS for the English version of the message resource file. In the sample JCL, the physical name of the message resource file is P390A.DMSGENUS.

5. To specify a control value to identify the MESSAGEFILE as DD:EXMSG, enter the following command:

```
//EXMSG DD DSN= <PDS>.VERMSG(MSGAFP2) ,DISP=SHR
```

All engine messages are written to this file. In the sample JCL, this statement equates it to the physical file MSGAFP2 in the PDS P390A.EXSTREAM.VERMSG.

6. To identify the data file for the application, enter the following command:

```
//BANKDATA DD DSN= <PDS>.VERDATA(BANKDATA) ,DISP=SHR
```

A value of DD:BANKDATA was entered as the name of the production file used by this application. In the sample JCL, the statement equates the logical name BANKDATA to the physical file BANKDATA in the PDS P390A.EXSTREAM.VERDATA.

7. To specify a control value to identify the PACKAGEFILE as the DD:PACKAGE, enter the following command:

```
//PACKAGE DD DSN= <PDS>.VERPACK(PACKAFP) ,DISP=SHR
```

The package file is the file that the engine uses to create output. In the example JCL, this statement equates it to the physical file PACKAFP in the PDS P390A.EXSTREAM.VERPACK.

8. To specify the control value to identify REPORTFILE as the DD:EXREPORT, enter the following command:

```
//EXREPORT DD DSN= <PDS>.VERRPT(RPTAFP2) ,DISP=SHR
```

All engine report information is written to this file. In the sample JCL, this statement equates it to the physical file RPTAFP2 in the PDS P390A.EXSTREAM.VERRPT.

9. To specify whether engine options are read from a control file, enter the following command:

```
//EXCONTROL DD DSN= <PDS>.VMSCTRL ,DISP=SHR
```

The engine options are read from this file when the switch -USECONTROL=YES is implemented. In the sample JCL, this statement equates the logical name DD:EXCONTROL to the physical file P390A.EXSTREAM.VMSCTRL.

10. To specify the tracking file, enter the following command:

```
//EXTRACK DD DSN= <PDS>.VERTRK(TRKAFP2),DISP=SHR
```

In z/OS, the default value for the tracking file is DD:EXTRACK. In the sample JCL, this statement equates it to the physical file TRKAFP2 in the PDS P390A.EXSTREAM.VERTRK.

11. To specify a control value to identify the OUTPUTFILE as the DD:EXOUTPUT, enter the following command:

```
//EXOUTPUT DD DSN= <PDS>.VERAFP(EXAFP2),DISP=SHR
```

The engine writes all the data destined for the output device to this file. In this example JCL, the output is an AFP print stream and the statement equates it to the physical file EXAFP2 in the PDS P390A.EXSTREAM.VERAFP.

When you select the output property **Used resources only** in any print stream that supports it, the engine uses temporary files to generate the output. On z/OS systems, you must allocate a temporary file (DD:TEMP) in your engine JCL. The space allocated for your temporary file must be at least as large as your main file.

## Configuring a Hierarchical File System for Output on the z/OS Production Engine

You can configure a Hierarchical File System (HFS) for output for the production engine on z/OS. HFS is supported for file output with the z/OS version used to run the engine. With HFS, the output file size and attributes resolve themselves. The Data Definition (DD) statement takes the following format:

```
//OUT1 DD PATH='/u/joeuser/BANK.pdf',  
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),  
// PATHMODE=(SIRWXU),PATHDISP=(KEEP,DELETE)
```

To	Do this
Specify the access and status of the data file	Use the PATHOPTS parameter.
Specify that the program can open file for writing	Use the OWRONLY parameter.
Specify that the system can create the file if it does not already exist	Use the OCREAT parameter.

To	Do this
Specify that the system is to truncate the file length to zero	Use the OTRUNC parameter.  You must verify the following conditions to be true before setting this parameter: <ul style="list-style-type: none"><li>• When the file specified on the PATH parameter exists</li><li>• When the file is a regular file</li><li>• When the file successfully opened with ORDWR or OWRONLY</li></ul>
Specify the file access attributes when the system is creating the HFS file named on the PATH parameter	Use the PATHMODE parameter.  Creating the file is specified by a PATHOPTS=OCREAT parameter. Use the SIRWXU permission to allow the file owner to do the following: <ul style="list-style-type: none"><li>• Read, write, and search, if the file is a directory.</li><li>• Read, write, and execute, for a file other than a directory.</li></ul>
Specify what happens to the file if the job ends normally or abnormally	Use the PATHDISP parameter.
Specify that the file should be kept	Use the KEEP parameter.  When using the KEEP parameter, you must verify the following: <ul style="list-style-type: none"><li>• When the step ends normally, KEEP is the first subparameter.</li><li>• When the step ends abnormally, KEEP is the second subparameter.</li></ul>
Specify that the file should be deleted	Use the DELETE parameter.  When using the DELETE parameter, verify the following: <ul style="list-style-type: none"><li>• When the step ends normally, DELETE is the first subparameter.</li><li>• When the step ends abnormally, DELETE is the second subparameter</li></ul>

For more information about output, see *Creating Output* in the Exstream Design and Production documentation.

## Configuring a JCL to Load and Run Multiple Package Files as VSAM Package Files

When you load multiple package files on a z/OS system, the engine tests each package file to verify if they are Virtual Storage Access Method (VSAM) package files. A VSAM package file is especially useful when running on a 32-bit engine and your applications contain large amounts of customer data.

Using VSAM package files improves the performance of the engine by allowing the index of available objects in memory to contain only the objects required for the current run. All package files must be VSAM files to improve the performance of the engine. Otherwise, the engine reads objects from all of the package files into memory and the index is populated for the engine run.

For more information about engine performance, see [“Optimizing Production Engine Performance” on page 88](#).

You can load a package file into either an ESDS or KSDS VSAM file. Use the following example JCLs as a guide to configuring and loading VSAM package files for an engine run.

JCL to configure the package file in an ESDS VSAM file:

```
//DELETE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
DELETE SVCS.TEST.TC36984.OUTPUT
DELETE SVCS.TEST.TC36984.PUB1ESDS
DELETE SVCS.TEST.TC36984.PUB2ESDS
//DEFINE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
/* DEFINE VSAM ESDS DATASET
DEFINE CLUSTER -
  (NAME(SVCS.TEST.TC36984.PUB1ESDS) -
   RECORDSIZE(80 32000) -
   NONINDEXED -
   VOLUMES(XBIG01)) -
  DATA -
   (CYLINDERS (2 1))
/*
```

JCL to configure the package file in a KSDS VSAM file:

```
//DELETE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
/* DELETE OLD DATASET
DELETE -
  EXSTREAM.KSDS
/*
//DEFINE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
/* DEFINE VSAM KSDS DATASET
DEFINE CLUSTER -
  (NAME(EXSTREAM.KSDS) -
   RECORDSIZE(80 32000) -
   VOLUMES(XBIG01)) -
  DATA -
   (KEYS(12 7) -
    CYLINDERS (200 200)) -
  INDEX -
   (TRACKS (5 5))
/*
```

JCL used to load the package file into either VSAM file:

```
VSAMRPRO
//REPRO   EXEC PGM=IDCAMS
//INPUT   DD DSN=QA.TC13811.PACKAGE(PKG4),DISP=SHR
//OUTPUT  DD DSN=EXSTREAM.ESDS.PKG1,DISP=SHR
//SYSPRINT DD SYSOUT=*
//*       REPRO
REPRO -
INFILE(INPUT) -
OUTFILE(OUTPUT)
/*
```

Use the following example JCL as a guide to configuring and running the engine using a VSAM package file:

```
//EBCEXE   EXEC PGM=ENGEXE,
//          PARM=' -USECONTROL=YES '
//STEPLIB DD DSN=EXSTREAM.VNN.LOAD,DISP=SHR
//DLMSGRES DD DSN=EXSTREAM.DMSGENUS,DISP=SHR
//EXMSGSG DD DSN=EXSTREAM.MESSAGES.OUT(MSG1),DISP=SHR
//EXCONTRL DD DSN=EXSTREAM.INPUT(CONTROL1),DISP=SHR
//EXTRACK DD DSN=EXSTREAM.OUTPUT(TRACK),DISP=SHR
//EXREPORT DD DSN=EXSTREAM.OUTPUT(REPORT),DISP=SHR
//EXPKG1   DD DSN=EXSTREAM.VSAM.PKG1,DISP=SHR
//IN30832  DD DSN=EXSTREAM.INPUT(IN30832),DISP=SHR
//IN30932  DD DSN=EXSTREAM.INPUT(IN30932),DISP=SHR
//PDF      DD DSN=EXSTREAM.PDF(PDF),DISP=SHR
```

## Improving z/OS Performance Using VSAM Clusters to Store Package Files

Various tuning options can be used to improve the performance of z/OS batch jobs when using VSAM clusters to store the package file. The best tuning method to use depends on the system software available to you.

For more information about improving overall engine performance, see [“Optimizing Production Engine Performance” on page 88](#).

You can use the following three methods to improve the performance of the engine when using VSAM clusters to store package files:

- Innovation Access Method (IAM)—You can use IAM (a competitor to VSAM) by making defining IAM as the OWNER in the DEFINE CLUSTER statement (Not all z/OS environments have access to IAM).

For example:

```
DEFINE CLUSTER ( NAME(EXSTREAM.VSAMTEST.ESDS) -  
OWNER($IAM)           - this makes it an IAM dataset  
STORCLAS(TEMP)         -  
NONINDEXED             -  
RECORDSIZE(80 32000)   ) -  
DATA ( TRACKS(600 50) ) \
```

- System Managed Buffering (SMB)—z/OS offers SMB, a feature of DFSSMS, as a technique to improve VSAM I/O. If you use the SMB option, you must allocate the VSAM dataset in a SMS dataclass that supports Extended Format and you must add the ACCBIAS parameter to the EXPACKAG DD statement. For example:

```
DEFINE CLUSTER (NAME(EXSTREAM.VSAMTEST.ESDS) -  
VOLUMES(TESTDA) -  
DATACLASS(XBIG) - this dataclass supports Extended Format  
NONINDEXED -  
RECORDSIZE(80 32000)   ) -  
DATA ( TRACKS(600 50)   )  
//PASS1 EXEC PGM=ENGEXE,  
...  
/* this is the DD for randomly reading the package during customer  
processing  
/* DO: SMB optimizes buffers management to direct access.  
//EXPAG DD DSN= EXSTREAM.VSAMTEST.ESDS,DISP=SHR, -  
//          AMP=('ACCBIA=DO')
```

**Note:** A z/OS system programmer can help you identify which data classes support the Extended Format. You can see if the dataset is in extended format by using LISTCAT ALL on the VSAM cluster and looking for Extended in the attribute section.

- Batch Local Shared Resources (BLSR)—You can use BLSR in the DD statement that references the VSAM package file. This is done when configuring the engine step. For example:

```
//PASS1 EXEC PGM=ENGEXE,  
...  
//ESDSEXT DD DSN= EXSTREAM.VSAMTEST.ESDS,  
// DISP=(OLD,DELETE)  
//EXPAG DD SUBSYS=(BLSR, 'DDNAME=ESDSEXT', 'BUFND=181')
```

In the previous example, EXPAG is the default DDNAME for the package file and ESDSEXT is a temporary DDNAME that references the actual VSAM package file.



## 6.3 Special Considerations for Running Multiple Instances of the Engine

To run multiple instances of the engine at the same time, whether in batch mode or real time, there are some considerations:

- Make sure an instance does not write to the same report or output file as another. You can usually do this by using the FILEMAP and VARSET switches.
- If you are running in batch mode or if you are running the engine using the On Demand module without the DDMESSAGE=APPEND switch, make sure each instance will not try to write the message file to the same place.

## 6.4 Understanding System Generated Files for the Production Engine

In addition to the output, you can also generate special files when the engine runs. You can generate message files and report files. Both types of system generated files can be used to troubleshoot issues encountered while running the engine.

You can specify engine message files in a text file format or XML file format.

For more information about configuring Exstream to generate system files, see [“Setting Engine Options” on page 11](#).

### 6.4.1 Generating Message Files While Processing Applications

Message files show you all the messages the engine generates while processing the application.

Four options are available when you create a message file. The four options are located in the **Message level** drop-down list on the **Run the Engine** dialog box. Each option controls how much detail is provided, relating to the severity of the messages reported.

From the **Message level** drop-down list, select from the following options:

- **All (info, warning, and error)**—All messages generated by the engine are reported. (This corresponds with choosing I for MESSAGELEVEL.)

- **Warning and error**—Only warning and error messages are reported in the message file. (This corresponds with choosing W for MESSAGELEVEL.)
- **Error only**—Only error messages are reported in the message file. (This corresponds with choosing E for MESSAGELEVEL.)
- **None**—No message file is generated.

The **Message level** options can also be controlled by using the MESSAGEFILE and MESSAGELEVEL engine switches.

## Changing the Encoding of Engine Messages for DBCS Applications

If you are working with DBCS applications, you might need to change the encoding of your engine messages. To change the encoding, use the -MSGENCODE=encodeType engine switch.

Some common choices for encodeType include the following:

- ASCII
- BIG5
- EBCDIC
- ISO88592
- JIPS
- JIS
- LATIN1
- LATIN2
- UTF16

**Note:** UTF16 is the default if no encoding is specified.

### 6.4.2 Generating Report Files to Show Details of Each Application Object

Report files show details about each object in the application.

Four options are available when you create a report file. The four options are located in the **Reporting level** drop-down list on the **Run the Engine** dialog box. Each option controls how much detail is provided, regarding the qualification and inclusion of objects for each customer.

The reporting level options available are:

- **None**—No report is generated. (This corresponds to choosing NONE for the REPORT engine switch.)
- **Selection summary**—A basic summary selection for the entire application is generated. (This corresponds to choosing SUMMARY for the REPORT engine switch.)
- **Customer selections**—A basic summary of the selections made for each customer is generated. (This corresponds to choosing CUSTOMER for the REPORT engine switch.)
- **Customer details**—A detailed report about all objects for each customer is generated. (This corresponds to choosing DETAIL for the REPORT engine switch.)

The **Reporting level** options can also be controlled by using the REPORTFILE and REPORT engine switches.

## Analyzing Objects at the Application Level With the Selection Summary Report

The **Selection summary** engine report provides the least amount of object data. It provides details at the application level of objects selected for the run. The following example shows an engine report file with **Selection summary** selected. In the following example, the customer driver file contains only one customer.

```
*****
OpenText Exstream Version 9.5.301
* Execution Begins: 9/27/2016   12:22:23
* Report File
(15)*****
Customers Processed:           1
Campaigns Selected:           0
Campaign Messages:            0
Other Messages:                0
Pages Processed:               1 (0 blank)
Sheets Produced: 1
```

## Analyzing Selections at the Customer Level With the Customer Selections Report

The **Customer selections** option provides a minimal amount of customer detail, similar to the **Selection summary**, but the selections are shown at a customer level, rather than an application level. The following example was generated using the same application as the previous example.

```
*****
*****
OpenText Exstream Version 9.5.301
* Execution Begins: 9/27/2016 11:00:41
* Report File (15)
*****
*****

* Run Selections: 1:
*****
***** COMPOSED PAGE LIST *****
**** COMPOSE INFO: Total pages: 1, Mktg: 0, Bus: 1, Inserts: 0, weight
0.150oz,
Language: English
Basic Automated 1, 1, 1 Front Page=Basic Table Papertype (8.5 x
11) Weight
( 0.150 oz)
***** QUALIFICATION REPORT *****
Qualified Document List (1):
1q, 1s Basic Automated (1 pages/messages)
1q, 1s Page (ordered): Basic Tabl
```

## Analyzing Object Information for Each Customer With the Customer Details Report

The **Customer details** option provides the most customer detail in the report files. This option generates information about each object used for each customer. The following sample report was generated using the same application previously used. Note that there is only one customer for this run.

Each value of the variables used for the customer is reported, as well as the values at specific times throughout the run. This is in addition to the qualification information already provided with lower levels of detail. This type of detail is useful for troubleshooting variables to ensure they reset properly.

```

*****
* OpenText Exstream Version 9.5.301
* Execution Begins: 9/27/2016 11:00:41
* Report File (15)
*****
*****
* Overview of File: Transactions
*****
* FORMAT: Record types
* The record types which have been mapped are:
*   0: (new customer)
*   C: (new customer)
*   T: (data record)
*****
*****
*           Variable Information
*****
SYS_CustomerEffectiveDate    Date        System    As-needed
SYS_LanguageCustomer        String      System      Cus-
tomer(Don't Compute)
SYS_LocaleCustomer          String      System      Cus-
tomer(Don't Compute)
SYS_CustInvalidDataLevel    Integer     System      Post-cus-
tomer(Don't Compute)
CustomerName                 String      File        As-needed
CustomerAddress1             String      File        As-needed
CustomerAddress2             String      File        As-needed
CustomerAddress3             String      File        As-needed
ApprovedCard                 String      File        As-needed
ChargeTransactionDate         Date        () File      As-needed
ChargeTransactionCity         String      () File      As-needed
ChargeTransactionDescription  String      () File      As-needed
ChargeAmount__               Currency    () File      As-needed
StatementBeginBal            Currency    File        As-needed
CreditCardNumber             String      () File      As-needed
StatementTotal                Currency    Formula     As-needed
***** Program Initialization
CustomerName                  ;
CustomerAddress1              ;
CustomerAddress2              ;
CustomerAddress3              ;
ApprovedCard                  ;
ChargeTransactionDate
ChargeTransactionCity
ChargeTransactionDescription
ChargeAmount__
StatementBeginBal              $0.00;
CreditCardNumber
StatementTotal                  $0.00;

```

```
***** Customer: 1 : Customer Data Read
CustomerName                Robert Stevens;
CustomerAddress1            100 Main Street;
CustomerAddress2            Apartment 104;
CustomerAddress3            Lexington, KY 40505;
ApprovedCard                ITSA Card Platinum;
ChargeTransactionDate        December 14, 2002; December
18, 2002; December 19, 2002; December 22, 2002; December 27, 2002;
ChargeTransactionCity        Louisville, KY; Louisville,
KY; Louisville, KY; Louisville, KY; Louisville, KY;
ChargeTransactionDescription  Outback Steakhouse; Yankee
Candle; Nordstrom's 33; Jiffy Lube; Kroger;
ChargeAmount__              $29.45; $84.76; $245.67;
$31.26; $52.25;
StatementBeginBal           $0.00;
CreditCardNumber            7512 2145 4517 9813
StatementTotal
$443.39;*****
*      Run Selections: 1:
*****
***** COMPOSED PAGE LIST *****
**** COMPOSE INFO: Total pages: 1, Mktg: 0, Bus: 1, Inserts: 0, weight
0.150oz,
Language: English
Basic Automated      1,   1,   1      Front Page=Basic Table Papertype (8.5 x
11) Weight
( 0.150 oz)
***** QUALIFICATION REPORT *****
Qualified Document List (1):
1q, 1s Basic Automated (1 pages/messages)
1q, 1s Page (ordered): Basic Table
***** Customer: 1 : Customer Processed
CustomerName                Robert Stevens;
CustomerAddress1            100 Main Street;
CustomerAddress2            Apartment 104;
CustomerAddress3            Lexington, KY 40505;
ApprovedCard                ITSA Card Platinum;
ChargeTransactionDate        December 14, 2002; December
18, 2002; December 19, 2002; December 22, 2002; December 27, 2002;
ChargeTransactionCity        Louisville, KY; Louisville,
KY; Louisville, KY; Louisville, KY; Louisville, KY;
ChargeTransactionDescription  Outback Steakhouse; Yankee
Candle; Nordstrom's 33; Jiffy Lube; Kroger;
ChargeAmount__              $29.45; $84.76; $245.67;
$31.26; $52.25;
StatementBeginBal           $0.00;
CreditCardNumber            7512 2145 4517 9813
StatementTotal
$443.39;
***** Program Completion
```

CustomerName	Robert Stevens;
CustomerAddress1	100 Main Street;
CustomerAddress2	Apartment 104;
CustomerAddress3	Lexington, KY 40505;
ApprovedCard	ITSA Card Platinum;
ChargeTransactionDate	December 14, 2002; December
	18, 2002; December 19, 2002; December 22, 2002; December 27, 2002;
ChargeTransactionCity	Louisville, KY; Louisville,
	KY; Louisville, KY; Louisville, KY; Louisville, KY;
ChargeTransactionDescription	Outback Steakhouse; Yankee
	Candle; Nordstrom's 33; Jiffy Lube; Kroger;
ChargeAmount__	\$29.45; \$84.76; \$245.67;
	\$31.26; \$52.25;
StatementBeginBal	\$0.00;
CreditCardNumber	7512 2145 4517 9813
StatementTotal	\$443.39;

## 6.5 Comparing and Viewing Exstream Batch Compare Files

You can compare and view ECF files in the production environment to contrast composed output from two different engine runs. The Exstream Batch Compare tool is intended for full comparisons of production output because it runs faster than the Visual Compare tool.

You can compare ECF files created from Exstream version 2.0 and later.

After generating the ECF files, you can compare the two files by running the Exstream Batch Compare tool with a batch control file. At a minimum, the control file must include the COMPARE and KEY switches. You can also add the following optional switches:

- BROWSEDIFF
- CONTROLFILE
- DIFFERENCEFILE
- DIFFTOLERANCE
- LINE\_TOLERANCE
- MESSAGEFILE
- POS\_TOLERANCE
- SIMPLEReturn

To execute the Exstream Batch Compare tool:

1. From the directory where the ECF files are located, enter the following:

```
CompareECF -CONTROLFILE=<filename>.
```

2. Press **ENTER**.

To open the difference file in Designer:

1. At the command prompt, enter the following:

```
"<file location of Designer>" -browsediff=<differencefile filename>
```

Parameters must be separated by carriage returns.

2. Press **ENTER**.

If you do not use the most recently opened database for the comparison, you are prompted you for the database when it launches Designer.

#### Exstream Batch Compare tool control file switches

Switch	Value	Default
-COMPARE=ECFfilename1,ECFfilename2	This switch specifies the viewer output files to compare to identify differences between two ECF files. The switch is required to run the Automated Output Compare tool.	No default
-CONTROLFILE=filename	This switch specifies the name of the control file to use for the application.	No default
-DIFFERENCEFILE=filename	This switch specifies a file containing a summary of differences. Use the BROWSEDIFF switch to automatically open the side file in Designer.	No default
-DIFFTOLERANCE=number	This switch specifies the number of pixels tolerated before acknowledging a difference in position or line thickness.	1



Exstream Batch Compare tool control file switches, continued

Switch	Value	Default
-KEY=value	<p>This switch identifies the text string of license key.</p> <ul style="list-style-type: none"> <li>• In Exstream 6.0 and later, the key is 155 characters with no spaces.</li> <li>• In Exstream versions 5.0 to 6.0, the key is 50 or 155 characters with no spaces.</li> <li>• In Exstream versions 3.5 to 5.0, the key is 50 characters with no spaces.</li> <li>• In Exstream 3.0, the key is 25 characters with no spaces.</li> </ul> <p>This switch, which is required in order to run the Exstream Batch Compare tool, overrides the key in the package file.</p>	No default
-LINE_TOLERANCE=number	This switch specifies the number of pixels tolerated before acknowledging a difference in line thickness.	1
-MESSAGEFILE=filename	This switch specifies the file name the engine uses to create a message reporting file.	<p>Default for PC and UNIX: ExstreamMessages.dat</p> <p>For z/OS: DD:MESSAGE</p>
-POS_TOLERANCE=number	This switch specifies the number of pixels tolerated before acknowledging a difference in position.	1
-SIMPLEReturn	This switch returns 1 for difference or 0 if no difference.	No default
-XMLMESSAGEFILE=filename	This switch specifies the XML file name the engine uses to create a message reporting file.	No default

## 6.5.1 Viewing and Comparing Applications Using Exstream Batch Compare

To specify a side file to open automatically in Designer, enter `-BROWSEDIFF=filename` at the command prompt. The two ECF files and the side file must reside in the same directory.

To open the difference file in Designer, enter:

"<file location of Designer>" `-browsediff=<differencefile filename>` at the command prompt and press **ENTER**. If you do not use the most recently opened database for the comparison, you are prompted you for the database when it launches Designer.

## Exstream Batch Compare Return Codes for the MESSAGEFILE Engine Switch

When you include the MESSAGEFILE engine switch, the engine returns an integer in the message file. Parse the integer to determine the return code(s).

Exstream Batch  
Compare tool return  
codes

Message	Integer
Invalid files or options	-1
Identical, no differences	0
Order changed	1
Position 1	32
Position 2	64
Content	128
Color	256
Angle	512
Border lines	1024
Text	2048
Images	4096
Fonts	8192
Lines	16384
Shapes	32768
Pages	524288
Customers	1048576

## 6.6 Engine Return Codes

The engine returns a code at the completion of every engine run. Engine return codes can initiate further action if a custom script is programmed to make it do so. For example, you can use the return code to interrupt production in case of a warning level error. Return codes are the same on all platforms. The following table describes the return code messages.

Engine return codes

Message	Message suffix	Return code
Normal exit with no errors or warnings	N/A	0
Informational	I—Informational	0
Normal exit with warnings	W—Warning	4
Normal exit with errors	E—Error	8
Fatal error—Program stop	S—Severe	12

The **Normal exit with errors**, return code 8, can also have warnings. The engine returns the maximum code value.