Oracle FLEXCUBE Direct Banking

UI Content Delivery Optimization Guide

Release 12.0.2.0.0

Part No. E50108-01

September 2013

ORACLE®

UI Content Delivery Optimization Guide

September 2013

Oracle Financial Services Software Limited
Oracle Park
Off Western Express Highway
Goregaon (East)
Mumbai, Maharashtra 400 063
India
Worldwide Inquiries:
Phone: +91 22 6718 3000
Fax:+91 22 6718 3001
www.oracle.com/financialservices/

CONTENTS

1. Preface

1.1. Intended Audience

This document is primarily targeted at

- Oracle FLEXCUBE Direct Banking Development Teams
- Oracle FLEXCUBE Direct Banking Implementation Teams
- Oracle FLEXCUBE Direct Banking Implementation Partners

1.2. Documentation Accessibility

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

1.3. Access to OFSS Support

https://flexsupp.oracle.com/

1.4. Structure

This document, termed “Oracle FLEXCUBE Direct Banking UI Content Delivery Optimization Guide”, provides the details of various optimization done in the web archive of Oracle FLEXCUBE Direct Banking

1.5. Related Information Sources

For more information on Oracle FLEXCUBE Direct Banking Release 12.0.2.0.0, refer to the following documents:

- Oracle FLEXCUBE Direct Banking Licensing Guide

2 INTRODUCTION

UI components like images, stylesheets and browser-scripts are a very important part of the application. These components are delivered to the browser over the network and are interpreted by the browser for rendering.

A given screen might use a large number of such components as part of its design. If these components are used *as-is* it can cause several issues such as:

- Large number of network requests
- Large payload (total data transferred for rendering a screen)
- Increased processing / wait time incurred by the browser

In order to mitigate any undesirable effects, the following optimization steps have been undertaken as part of the product packaging activity:

- Scripts have been minified
- Related script files have been combined into a single script file
- Stylesheets have been minified
- Image files have been converted to their Base64 representation and the result has been embedded into stylesheets
- Static and dynamic content transferred from the application is compressed using GZIP

As a result of these optimization steps, the UI content is not directly modifiable on an *as-deployed* basis. In order to update such content, one needs to execute the applicable sequence of steps as provided in this manual.

3 USE CASES

A deployed WAR contains UI elements in the following form:

- combined and minified script files
- Images converted to their Base64 representation and embedded into stylesheets
- Minified stylesheets

Apart from this, you have been supplied with the source files for the following components:

- original (non-minified) script files
- original (non-minified) stylesheets with physical locations of image files specified
- original image files

Below examples will better explain the changes that have been done:

Example 1: Java script file jsdir/common.js:

Before 12.0.1 release	Changes in optimized WAR of 12.0.1 release
jsdir/common.js – This file is readable and can be directly debugged or modified	jsdir/template/common.js – This file is readable although it need to be copied to main folder for debugging jsdir/common.js – This file is minified version of the file in template folder. This is not readable.

Example 2: Style sheet file css/cmn/taskbar.css:

Before 12.0.1 release	Changes in optimized WAR of 12.0.1 release
css/cmn/taskbar.css – This file is readable and can be directly debugged or modified. This file contains some image references.	css/cmn/template/taskbar.css – This file is readable although it need to be copied to main folder for debugging css/cmn/taskbar.css – This file is minified version of the file in template folder. This is not readable. This file will only be used by IE6 and IE7 browsers. css/cmn/taskbar.uri.css – This file contains all image references replaced by Data URI. This file is also in minified form and is not readable. This file is used by all browsers except IE6 and IE7.

Here are some common instructions for making any changes to the deployed UI components:

- The JS and CSS files referred from browser will be minified format and hence not readable. So copy the file from template folder for modification or debugging, and place the updated file back in template folder once the work is done.
- The use cases in subsequent sections cover specific scenarios.
- Always promptly run the optimizer utility and rebuild the WAR whenever any changes occur. The utility picks up files from template folders, optimizes them and puts them in the main folder (from where they will be used by browser)
- Details of various options of optimizer utility are available in subsequent sections.

To make any changes to these UI components deployed in the WAR, you may be required to take one or more of the approaches specified below.

A. ADDING A NEW SCRIPT

When adding a single new script to a screen (XSL) that uses no other scripts:

- proceed to include the script in the conventional manner

When adding one or more new script(s) to a screen (XSL) that uses one or more other individual scripts:

- combine the new and existing individual scripts using the Optimizer utility as per the instructions specified in 'Combining Files' section below
- Add only the resulting combined script file to the screen
- Comment out references to the old script files
- Add a reference to the newly added file in the commented out section

B. DEBUGGING OR MODIFYING AN EXISTING SCRIPT

In order to debug a script the following steps need to be performed:

- Look up the *original source* of the script that needs to be debugged in template folder
- Replace the minified version in the WAR with the original source
- Perform debugging tasks and updates
- Once the changes are final, copy the updated file to template folder
- Rebuild the WAR file
- Redeploy the updated WAR

Script minification is performed as part of the build process

C. ADDING A NEW STYLESHEET

When adding a new stylesheet to the set of stylesheets:

- Follow the instructions given in section 'Converting a single stylesheet' mode described below

D. DEBUGGING OR MODIFYING AN EXISTING STYLESHEET

In order to debug a stylesheet, the following steps need to be performed:

- Look up the *original source* of the stylesheet that needs to be debugged in template folder
- Replace the minified version in the WAR with the original source
- Replace the .uri.css version with original source

- Perform debugging tasks and updates
- Once the changes are final, copy the updated file to template folder
- Rebuild the WAR file
- Redeploy the updated WAR

Data URI conversion and minification is performed as part of the build process

E. ADDING ONE OR MORE NEW IMAGES

When adding one or more new image(s) to the set of images:

- Never add an <img> tag in any XSL. Add a or as appropriate and specify the image using background-image property of CSS, and appropriate selectors

Rebuild the WAR file. Data URI conversion and minification of CSS is performed as part of the build process.

4 UI CONTENT OPTIMIZER UTILITY

A. INTRODUCTION

The UI content optimizer utility performs various optimization tasks that include

- JS and CSS minification
- Data URI conversion
- Combining of files
- Bulk Data URI conversion for menu icons

In addition to above, this utility provides an option to do above optimization tasks on the WAR using a single command. Subsequent sections provide more details on each of these activities. The following standard terms are used throughout this section:

BASE_LOCATION	Refers to the temporary working folder to which the contents of the currently deployed WAR file have been extracted. This is supplied as a run-time argument
CSS_HOME	Specifies a path relative to BASE_LOCATION, that refers to the top-level folder under which all stylesheets have been placed
ENTITY	Specifies entity for which this tool should to be run
BATCH_CONFIG	Specifies property file required during build process for proper execution of the tool
EXCLUDE_SCRIPT_FILES	Specifies list of script files which needs to be excluded during minification process.
MENUICONS_CSS_PATH	Specifies path where the output stylesheet file (menu-icons.css) should be placed after base 64 conversion for menu icons.
MENUICONS_HOME	Specifies a path relative to BASE_LOCATION, that refers to the top-level folder under which the set of image files used as menu icons have been placed

B. ONE-CLICK BUILD PROCESS

This mode is used to perform all the activities required during build process. It performs the following tasks in the given order:

1. Bulk convert all menu icons to use data URI
2. Combine scripts as per batch configuration file
3. Bulk convert all stylesheets to use data URI
4. Minify all stylesheets
5. Minify all scripts

Invoke the optimizer utility in the following manner

```
$ java JFUIContentOptimizer -build -properties <BATCH_CONFIG> <ENTITY>
```

C. BULK CONVERT ALL STYLESHEETS

This mode will convert all stylesheets placed at the location <BASE_LOCATION + CSS_HOME> and all its sub-directories.

This mode can be used for converting non-optimized stylesheets to optimized ones, as well as for updating optimized stylesheets to reflect recent modifications.

Using this, the image URLs present in the input non-optimized stylesheets are converted to data URIs having the Base64 representations of the image files that are being referred to.

I. STEPS

1. Unzip the deployed WAR to <BASE_LOCATION>
2. Invoke the optimizer utility in the following manner

```
$ java JFUIContentOptimizer -stylesheet -bulk <BASE_LOCATION>
```

3. Repackage the unzipped (and subsequently updated) contents back to a WAR file
4. Redeploy the updated WAR file

II. POINTS TO NOTE

- The optimized stylsheets are generated and placed in the parent folder of template directory where each original stylesheet is present. The optimized stylesheets are suffixed with.uri in their file name
- The original stylesheets are left as is, for backward compatibility with Internet Explorer versions 6.x and 7.x.

D. CONVERT SINGLE STYLESHEET

This mode will convert a single stylesheet to its optimized representation.

Using this, the image URLs present in the input non-optimized stylesheet are converted to data URIs having Base64 representations of the image files that are being referred to.

I. STEPS

1. Unzip the deployed WAR to <BASE_LOCATION>
2. Place the new (input) stylesheet at the intended location within <BASE_LOCATION + CSS_HOME>. This location, is referred to as <SOURCE_FILE> and includes the file name itself.
3. Place any new or updated image files at their intended location
4. Invoke the optimizer utility in the following manner

```
$ java JFUIContentOptimizer -stylesheet -single <SOURCE_FILE>
<DESTINATION_FILE>
```

5. DESTINATION_FILE should be the path where the output stylesheet file should be generated after data URI conversion.
6. Repackage the unzipped (and subsequently updated) contents back to a WAR file
7. Redeploy the updated WAR file

II. POINTS TO NOTE

- It is important that the output stylesheet and the associated new or updated images be present at their intended locations for the relative paths given in the stylesheet to be realizable

E. BULK CONVERT ALL MENU ICONS

The Base64 representation of the entire menu icon set (in the given file format) found in <BASE_LOCATION + MENUICONS_HOME> is present in the stylsheet *menu-icons.css* found at <BASE_LOCATION + CSS_HOME>/cmn/template. Template folder under cmn should already exist.

If one or several of the icons is/are updated, the stylesheet file needs to be updated.

The optimizer performs the following functions:

1. Picks up all the images found at the location <BASE_LOCATION + MENUICONS_HOME>
2. Converts each of those images to its Base64 representation
3. Creates a finished CSS file that contains data URI references to the Base64 representations, set as the background-image property for the required selectors (HTML div class names)
4. Saves the CSS file at <BASE_LOCATION + CSS_HOME>/cmn/template with the name menu-icons.css

I. STEPS

1. Unzip the deployed WAR to a temporary folder. This folder will be referred to as <BASE_LOCATION>
2. Decide the file type (GIF, PNG, etc) of the menu-icons to convert. This utility requires that all the menu icons must be in the same file format <FILE_EXTN>
3. Invoke the optimizer utility in the following manner

```
 $ java JFUIContentOptimizer -image -bulk -menuicons <BASE_LOCATION>
<FILE_EXTN> <BATCH_CONFIG>
```

4. The output file menu-icons.css is generated and placed in <BASE_LOCATION + CSS_HOME>/cmn/template

II. POINTS TO NOTE

- Avoid using the JPEG format for image icons as they tend to have a very large size as compared to GIF or PNG for computer-generated graphics such as icons

F. BULK CONVERT MULTIPLE IMAGES

This mode is useful when:

- adding several new images to the collection

I. STEPS

1. Determine the file name extension of the images to be converted. This will be the <IMAGE_FILE_EXTN>
2. Specify the complete path of the location that contains the source images to be converted. This location will be <SOURCE_LOCATION>
3. Invoke the optimizer utility in the following manner

```
$ java JFUIContentOptimizer -image -bulk <SOURCE_LOCATION> <IMAGE_FILE_EXTN>
```

4. The result is saved in a .txt file having the same name as the source image folder, and in the same location as the source image file. If SOURCE_LOCATION is a drive then .txt file name will be temp.txt.

II. POINTS TO NOTE

- Only images whose file-name extensions match <IMAGE_FILE_EXTN> will be processed. Any other files in the <SOURCE_LOCATION> will be ignored
- The result is saved in a plain-text file as a CSS-style declaration that contains the Base64 representation in the background-image property
- The url (…) part must be copied to the desired location within the target stylesheet

G. CONVERT SINGLE IMAGE

This mode is useful when:

- adding a new image to a stylesheet

I. STEPS

1. Determine the location of the source image. This location will be the <SOURCE_IMAGE_LOCATION> and includes the file-name of the image
2. Invoke the optimizer utility in the following manner

```
$ java JFUIContentOptimizer -image -single <SOURCE_IMAGE>
```

3. The result is saved in a .txt file having the same name as the source image file, and in the same location as the source image file

II. POINTS TO NOTE

- The result is saved in a plain-text file as a CSS-style declaration that contains the Base64 representation in the background-image property
- The url (…) part must be copied to the desired location within the target stylesheet

H. COMBINE FILES - BY FOLDER

This mode is used to:

- combine two or more script files into a single file
- combine two or more stylesheets into a single file

I. STEPS

1. Decide the type (file-name extension) of files that you would like to combine. The desired file extension will be referred to as <FILE_EXTN>.
2. Place all the desired files into a temporary location. The complete path to this location will be referred to as <COMBINE_LOCATION>. This location should be given the name that is the desired file name of the combined script file
3. Invoke the optimizer utility in the following manner:

```
$ java JFUIContentOptimizer -combine <FILE_EXTN> <COMBINE_LOCATION>
```

4. The resulting combined file with the extension <FILE_EXTN> is generated and placed in the same location as that of <COMBINE_LOCATION>.

II. POINTS TO NOTE

- This mode combines all the available files having extension <FILE_EXTN>
- Files are combined in alphabetical order of file-name
- If certain files need to be prioritized over others during the process, please use batch mode as outlined in the next section

I. COMBINE FILES – BY BATCH CONFIGURATION FILE

This mode is used to combine files when:

- required files are present in different folders
- a specific (non-alphabetical) combine order of files is required

I. STEPS

1. Determine the type of files that you would like to combine. The file name extension of such files will be the <FILE_EXTN>
2. Create a new plain-text file
3. Save the file with the same name that you would like the combined file to have, and with an extension of .txt . This file is called as the batch file and the location where you will save this file (including the file name itself) will be the <BATCH_FILE_PATH>
4. Copy the full path of each file that you wish to include in the combined file. Each path must be on a new line
5. Invoke the optimizer utility in the following manner:

```
$ java JFUIContentOptimizer -combine <FILE_EXTN> <BATCH_FILE_LOCATION>
```

6. The resulting combined script file having the same name as the batch file, and the extension <FILE_EXTN> is generated and placed in the same folder as the batch file

II. POINTS TO NOTE

- Files are combined in the same order in which they are listed in the batch configuration file
- All files included in the batch, must be of the same type (scripts, stylesheets, etc). Do not mix different file types. If file types are mixed, only those files from the batch configuration that match the specified extension will be processed.
- Batch files with extensions other than .txt are also accepted. However, batch files with the extension <FILE_EXTN> will not be processed

J. MINIFY BULK STYLESHEET FILES

This mode is useful when:

- All stylesheet files present in the application needs to be minified.

I. STEPS

1. Check the BASE_PATH of the application in the <BATCH_CONFIG> file.
2. Next invoke the optimizer utility in the following manner:

```
$ java JFUIContentOptimizer -minify -bulk -CSS <BATCH_CONFIG>
```

3. Tool will place the minified stylesheet file in parent folder of template directory.

II. POINTS TO NOTE

- Tool will read stylesheet files present in only template directories.
- The original file in template directory will be unmodified.
- Instead the minified version will be available at parent directory of template folder.

K. MINIFY BULK SCRIPT FILES

This mode is useful when:

- All Script files present in the application needs to be minified.

I. STEPS

1. Check the BASE_PATH of the application in the <BATCH_CONFIG> file.
2. Next invoke the optimizer utility in the following manner:

```
$ java JFUIContentOptimizer –minify –bulk –JS <BATCH_CONFIG>
```

3. Tool will place the minified script file in parent folder of template directory.

II. POINTS TO NOTE

- Tool will read script files present in only template directories.
- The original file in template directory will be unmodified.
- Instead the minified version will be available at parent directory of template folder.

L. MINIFY SINGLE FILE

This mode is useful when:

- A single script or stylesheet file in the application needs to be minified.

I. STEPS

1. Invoke the optimizer utility in the following manner:

```
$ java JFUIContentOptimizer –minify –single <SOURCE_FILE> <DESTINATION_FILE>
```

2. <DESTINATION_FILE> will be the minified version of <SOURCE_FILE> file.

II. POINTS TO NOTE

- The original file <SOURCE_FILE> will be unmodified.

Instead the minified version will be available in <DESTINATION_FILE>.

5 A NOTE ON GZIP COMPRESSION

GZIP is a standard data compression mechanism that is supported by most of the browsers. The compression can be enabled in two different ways.

1. By default, a response filter is configured for the web archives in web.xml. This filter class is provided as part of the Kernel JAR
2. Alternatively, in implementation setup, typically there will be a Web Server (such as Apache HTTP server) in front of the application server. Gzipping of the data can be configured in this web server. If this approach is taken, then the response filter configuration should be removed from all web archives.

Please note that the above two approaches should be mutually exclusive, that is the compression must be done by one of the above mechanisms (and not both of them).