Oracle Utilities Cloud Services

Integration Guide

For 20B Releases

F32565-01

August 2020

Oracle Utilities Customer Cloud Services 20B Integration Guide

Contents

Chapter 1

Introduction

This document describes how Oracle Utilities cloud service implementations can integrate with other systems, including:

- File-based integration
- Connecting to an external printer for receipt printing

Chapter 2

File-Based Integration

The next three chapters describe how implementations can integrate and exchange information from Oracle Utilities Cloud Services to other applications and vice versa through file based integration. File upload and download processes are used by some implementation for data connect, payment upload, letters extract, financial extracts, other business processes.

Oracle Utilities Cloud Services can exchange data files from one application to another by:

- Directly accessing Oracle Cloud Object Storage
- Integrating with Oracle Integration Cloud.

  For more information about this approach, refer to the Oracle Integration Cloud documentation at https://docs.oracle.com/en/cloud/paas/integration-cloud/

These chapters provides information on how Oracle Utilities Cloud Services, specifically Oracle Utilities Customer Cloud Service (CCS), access data files from Oracle Cloud Object Storage, including:

- Chapter 3: Object Storage Connection Management
- Chapter 4: File Export Sample Implementation
- Chapter 5: File Import Sample Implementation

Chapter 3

Object Storage Connection Management

This chapter outlines how to manage connections between Oracle Utilities cloud services and Oracle Object Storage, including:

- Oracle Object Storage Setup
- Oracle Utilities Cloud Service Configuration for Object Storage Connection
- Register API Key to Oracle Cloud Object Storage

Oracle Object Storage Setup

Before initiating a file transfer from an Oracle Utilities cloud service to Oracle Cloud Object storage or vice versa, you must first make sure that the basic administration tasks in Oracle Cloud Infrastructure related to Object Storage have been completed properly, and that the compartments and buckets where the import and export files are stored have been setup.

For more information on Oracle Cloud Object Storage setup for Oracle Utilities Cloud Services, including Oracle Utilities Customer Cloud Service (CCS), please see the *Oracle Utilities Cloud Services Object Storage Setup Guide*.

Oracle Utilities Cloud Service Configuration for Object Storage Connection

Authentication and connection between the Oracle Utilities cloud service and Object Storage enables batch processes to import and export files from and to Object Storage locations. Setting up this authentication requires the following in your Oracle Utilities cloud service:

- Creating API Keys
- Creating An Object Storage Connection

Refer to the *Oracle Utilities Cloud Services Object Storage Setup Guide* for details concerning setting up Keys and Key Rings, an object storage connection configuration, and registering the API key.

> **Note**: You can use the same API Keys and Object Storage Connection setup for both import and export process.

Creating API Keys

Create a key ring for the cloud service environment. The key ring should be active and should have a set of private/public encryption key pairs. This key ring will be included in the Object Storage Connection Configuration.

Creating Key Rings and Pairs

Authentication between the Oracle Utilities cloud service and Oracle Object Storage requires an API signature key. See **API Key Management** in the *Oracle Utilities Cloud Services Object Storage Setup* Guide for more information.

API key rings and key pairs are maintained in the **Key Ring** portal in the cloud service. This portal contains the following zones:

- **Key Ring**: Displays basic information about the key ring
- **Key Pairs**: Displays a list of key pairs for the current key ring

Key rings are defined by the following:

- **Key Ring**: A unique code for the key ring
- **Key Ring Class**: Signature (default)

- **Status**: The current status of the key ring.

  Note: Key pairs can only be generated for Active key rings. Once a key ring has been deactivated, you can no longer create key pairs for that ring.

- **Description**: A name for the key ring (this will be referenced in the File Location extendable lookup, see below)

Once the key ring is created, you need to generate and activate the key pair. Click **Generate Key** to generate a key pair for the key ring.

Key Pairs are defined by the following:

- **Sequence**: The sequence of the key pair (the order in which the key pair was created)
- **Creation Date/Time**: The date/time when the key pair was created
- **Key Status**: The current status of the key pair. Key pairs are inactive when first created.
- **Public Key**: Click **View** to open a dialog box containing the public key.
- **Action**: Click **Activate** to activate an inactive key pair.

Click **Activate** in the **Actions** column in the **Key Pairs** zone. A dialog box opens displaying the following message: "Warning(s): Activating a key assumes that you have already registered the public key with the appropriate third parties. Press **Cancel** to abort." Click **OK** to activate the key. The **Key Status** column will change to "Active".

Note: Be sure to register the API Key with Object Storage by copying the public key to Oracle Identification and Access Management. To copy the public key, click **View** in the **Actions** column in the **Key Pairs** zone, and select and copy the text in the **View Public Key** dialog box. Refer to **Register API Key to Oracle Cloud Object Storage** on page 3-4 for more information.

Creating An Object Storage Connection

Create an object storage connection via the File Storage Configuration extendable lookup (F1-FileStorage). This defines the Object Storage location where the files will be stored.

Creating File Storage Extendable Lookup Values

Apart from the authentication, the cloud service also needs information about the Object Storage locations to be used. Object Storage locations are defined by values in the File Storage Configuration (F1-FileStorage) extendable lookup. These file storage configurations will be referenced by the batch processes that will import or export records.

Values for the File Storage Configuration extendable lookup are defined by the following:

- **Value**: A unique code for the extendable lookup value. This value will be referenced as a batch control parameter value.
- **Description**: A description of the extendable lookup value
- **Status**: The current status of the value. Select "Active".
- **File Storage Details**: This section defines details for the object storage location, including:

- **File Adapter**: The type of file adapter for the location. Select “Oracle Cloud Object Storage”.
- **User**: The user Oracle Cloud ID (ODIC) for the object storage location
- **Tenancy**: The tenancy Oracle Cloud ID (ODIC) for the object storage location
- **Compartment**: The compartment Oracle Cloud ID (ODIC) for the object storage location
- **Namespace**: The namespace for the object storage location
- **Key Ring**: The Key Ring you created earlier
- **Region**: The region of the object storage tenancy for the connection (Values for this field are defined in the F1_REGION_FLG lookup.

Refer to **External File Storage** in the *Oracle Utilities Application Framework Administrative User Guide* and the *Oracle Utilities Cloud Services Object Storage Setup Guide* for more information.

Register API Key to Oracle Cloud Object Storage

Once the key ring and key pair have been created in the Oracle Utilities cloud service, copy the public key from the key pair and add the public key to the **User API Key** in Oracle Identification and Access Management (IAM) See the **User API Keys** section in the **Security and Access Management** section of the **Managing Object Storage** chapter in the *Oracle Cloud Infrastructure Services* documentation.

Chapter 4

File Export Sample Implementation

This chapter provides an example of implementing a file export process, which includes:

- Creating a File Export Batch Process
- Configuring the Export Process

Creating a File Export Batch Process

Oracle Utilities cloud services provide a way to extract system information into a file using a file-export batch process. This process can be configured to export and store extracted files in an Object storage location.

Oracle Utilities cloud services provide a sample Batch Control (F1-PDBEX) which supports file export functionality and which can be used as 'template' to implement custom file export functionality as needed by specific implementations. Along with this out of the box batch control, cloud services also provide related objects (such as scripts.) that also can be used as templates while creating custom export processes.

Please note that the main data extraction logic lies within the script as described below. Customer can look at and copy the F1-GenProcEx script to implement their own data extraction logic.

This guide will follow a "bottom up" approach regarding the creation and configuration of cloud service data and objects to facilitate the file export process. The first step is to create a data area and script using that data area. Other objects for the file export will be created next.

For this sample implementation, we will export Premise information from Customer Cloud Service to Object Storage. This involves creating two algorithms (one for file export and one for selecting records to export) and a batch control.

To create a File Export Algorithm:

1. Create a data area that defines the schema for holding premise information.
2. Create a Plug-in script with the *Batch Control - Process Record* Algorithm Entity. This script may be based on the F1-GenProcEx script and modified as needed. Make sure to use the data area created above to hold premise information.

   The script will be used by the algorithm that will perform the file export.
3. Create an Algorithm Type similar to F1-GENPROCEX, based on the plug-in script created in the last step.
4. Create an algorithm based on the algorithm type.

To create a record selection algorithm:

1. Duplicate the F1-GENPROCSR algorithm to create a custom algorithm that will be used for selecting records.
2. Update the algorithm’s parameters.
   - Use the **SQL** parameter name to define the new query for retrieving records from the cloud service database.
   - Keep the **Batch Strategy** parameter as *THDS*.
   - Define the key field on the query under the **Key Field** parameter.

To create a batch control:

1. Duplicate the F1-PDBEX batch control to create a new batch control. Navigate to **Algorithms** tab on the batch control and replace the existing algorithms with the file export and record selection algorithms created above.
2. This newly created batch control contains parameters for file storage that must be modified as described in the following section.

Configuring the Export Process

This section describes the setup needed to enable export processing, including establishing communication between the Oracle Utilities cloud service and Object Storage, configuring batch parameters, and testing the export process.

- Setting Up Communication Between Cloud Service and Object Storage
- Configuring File Export Batch Parameters
- Testing the Export Process

Setting Up Communication Between Cloud Service and Object Storage

The export process requires authentication and communication between the Oracle Utilities cloud service and Object Storage. See Chapter 3: Object Storage Connection Management for more information about setting up this communication and authentication.

Configuring File Export Batch Parameters

The next step is to configure the following parameters of the file export batch control created earlier:

- **File Name**: the name of the exported file
- **File Path**: the path to file location in Object Storage. The format for the path is as follows:

  file-storage://<File Location>/<Bucket>

  where:

  - **<File-Location>**: The File Storage Configuration extendable lookup value defined for that file. This will include the compartment identification.
  - **<Bucket>**: The object storage bucket in the compartment that is defined as part of the File Storage Configuration extendable lookup value.

  For example, the "File-Export" bucket in a compartment that is referenced in the "AB-Export" File Storage Configuration extendable lookup value can be referenced as:

  "file-storage://AB-Export/File-Export"

- **Other Parameters**: The batch control supports other optional parameters, including:

  - **XML Root Name**: used to declare the name of root-node in generated xml (when exporting files in xml format)
  - **File Delimiter**: used to define the delimiter used in delimited files

  Refer to the batch control for information about other parameters.

Testing the Export Process

The last step is to test the export process.

To test the export process:

1. Run the file export batch process.
2. Navigate to the **Batch Run Tree** portal to verify the job has run successfully.
3. Navigate to targeted bucket inside Object Storage and verify the exported file.

Chapter 5

File Import Sample Implementation

Oracle Utilities cloud services include a batch control that can be used as a template for creating batch controls to import data from a file to the application. This template batch control is called Plug-in Driven File Upload Template (F1-PDUPL).

To use this template, an implementation can duplicate the F1-PDUPL batch control and provide the required algorithm for the "File Upload" system event. The algorithm associated with the batch control is responsible for using provided APIs to read the content of the file and store the data in appropriate table(s) such as a staging table or the FACT table. (we will use the FACT table in this sample).

The plug-in scripts written to implement this type of algorithm must use the Groovy script engine version as the APIs are not accessible using the XPath scripting language. The sample plug-in scripts provided illustrate using the various available APIs to upload a flat file, xml file or a delimited file. Implementation can write their own plug-in scripts to handle their specific file upload needs.

The following steps summarize how to implement a new file import background process:

- Identifying Upload File Content Data
- Uploading File to Oracle Cloud Object Storage
- Creating a File Import Batch Process
- Configuring the Import Process

For more information on how to use Plug-in Driven background processing for import and upload, refer to the following section in the *Oracle Utilities Application Framework Administrative User Guide*:

Background Processes

Understanding Background Processes

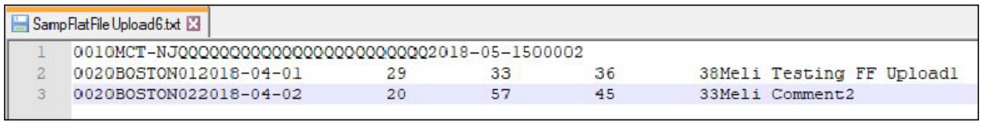
Plug-in Driven Background Processes

Uploading Records

Identifying Upload File Content Data

An important step in creating an import process is to define the format of the data files you plan to import, identify the different values in the file and map the data to fields in one or more appropriate tables in the application.

For example, the SampFlatFileUpload6.txt file has the following content



```
0010MCT-NJQQQQQQQQQQQQQQQQQQQQQQQQ2018-05-1500002
0020BOSTON012018-04-01        29        33        36        38Meli Testing FF Upload1
0020BOSTON022018-04-02        20        57        45        33Meli Comment2
```

This sample data contains 'degree day' data, including a header record and two data records.

The header record contains the following components (the length of the field is shown in parenthesis):

- Record Type (4) (Value: 0010)
- Source (30) (Value: MCT-NJ)
- Date Transmitted (10) (Value: 2018-05-15)
- Number of Records (5) (Value 00002)

Individual data records have the following components:

- Record Type (4) (Value: 0020)
- Area (8) (Value: BOSTON01, BOSTON02)
- Degree Date (10) (Value: 2018-04-01, 2018-04-02)
- Degree Day (10) (Value: 29, 20)
- Minimum Temperature (10) (Value: 33, 57)
- Average Temperature (10) (Value: 36, 45)
- Maximum Temperature (10) (Value: 38, 33)
- Comments (254) (Value: Meli Testing FF Upload..., Meli Comment2)

Uploading File to Oracle Cloud Object Storage

Once you have a sample file in the correct format, you need to upload the file to the location in Object Storage from where you plan to upload and import your data.

Creating a File Import Batch Process

Creating a file import process involves creating the following components in the Oracle Utilities cloud service:

- Plug-In Script
- Algorithm Type and Algorithm
- File Upload Batch Control

Plug-In Script

The first step is to create a plug-in script that will process the data in the upload file. This plug-in script should be created for the "Batch Control - File Upload" **Algorithm Entity**.

You can use sample plug-in scripts provided or create a new plug-in script with logic required for reading the record and identifying each record detail to properly create the insert statements for storing the data in the appropriate application tables.

The sample plug-in scripts provided illustrate how to call the supplied APIs for processing different types of source data including fixed position, comma delimited, and XML formats.

Sample Plug-in Scripts for Algorithm Entity: Batch Control - File Upload

Plug-In Script	Description
F1UplSmplFlt	Sample Flat File Upload Script
F1UplSmplDlm	Sample Delimited File Upload
F1UplSmplXML	Sample XML File Upload

Note: The F1UplSmplFlt sample script is designed to work with the sample data above.

Algorithm Type and Algorithm

The next step is to create an Algorithm Type and Algorithm that use the plug-in script you created above. In this sample file import implementation, the source data uses the fixed position format, so the Algorithm Type and Algorithm should use the F1UplSmplFlt script.

To create a new algorithm:

- Create an Algorithm Type using the F1UplSmplFlt plug-in script.
- Create a corresponding Algorithm for the Algorithm Type created above.

File Upload Batch Control

The last part of the cloud service configuration is to create a batch control that will use the new algorithm to import the data. You can create a new batch control by duplicating a template batch control and reference the new algorithm. The base product includes the Plug-in Driven File Upload (F1-PDUPL) batch control which can be used as a template.

To create a new batch control:

1. Duplicate the F1-PDUPL sample batch control to create your own batch control.
2. In the **Algorithms** tab, define the algorithm created above for the "File Upload" **System Event**.
3. Define default parameters for the batch control, if required.

Configuring the Import Process

This section describes the setup needed to enable file import processing, including establishing communication between the Oracle Utilities cloud service and Object Storage, configuring batch parameters, and testing the process.

The following steps will enable file import batch processing to run and import the data from the import file to the appropriate application tables in the Oracle Utilities cloud service:

- Setting Up Communication Between Cloud Service and Object Storage
- Configuring File Import Batch Controls
- Testing the Import Process

Setting Up Communication Between Cloud Service and Object Storage

The export process requires authentication and communication between the Oracle Utilities cloud service and Object Storage. See Chapter 3: Object Storage Connection Management for more information about setting up this communication and authentication.

Configuring File Import Batch Controls

The next step is to configure the following parameters of the file import batch control created earlier:

- **File Name**: The name of the file to import.
- **File Path**: the path to the file location in Object Storage where import files will be located. The format for the path is as follows:

  file-storage://<File Location>/<Bucket>

  where:

  - **<File-Location>**: The File Storage Configuration extendable lookup value defined for that file location. This will include the compartment identification.
  - **<Bucket>**: The object storage bucket in the compartment that is defined as part of the File Storage Configuration extendable lookup value.

  For example, the "File-Import" bucket in a compartment that is referenced in the "INT-UPLOAD" File Storage Configuration extendable lookup value can be referenced as:

  file-storage://INT-UPLOAD/File-Import
- Refer to the batch control for information about other parameters.

Testing the Import Process

The last step is to test the import process using the sample data.

To test the import process:

1. Run the file import batch process.
2. Navigate to the **Batch Run Tree** portal and make sure the batch process ran successfully.
3. Check that the records have been added to the FACT table.

Chapter 6

Customer Cloud Service Receipt Printing

This chapter describes how to configure Oracle Utilities Customer Cloud Service to support integration with a Point Of Sale (POS) printer for printing of receipts related to the following payment transactions:

- Payment Event
- Payment Event Quick Add
- Payment Quick Add

Refer to the Oracle Utilities Customer Cloud Service *Business User Guide* for more information about these payment transactions.

Configuration to support this functionality includes:

- Printer Installation
- Oracle Utilities Customer Cloud Service Configuration

Notes:

- The instructions in this document are based on a specific sample printer, the Epson TM-H6000IV-DT Series.

Printer Installation

This implementation requires installation of a Javascript printer library and SDK on a server that is accessible from the user's web browser (the connection to the printer will be directly from the web browser rather than from the application server).

For example, when using the sample Epson TM-H6000IV-DT Series printer, the following Javascript library would be installed on the server:

```
epos-x.x.x.js
```

Note: If the Oracle Utilities application or cloud service is using HTTPS, the Javascript library should be also be served over HTTPS to avoid mixed-content errors. SSL should be enabled on the printer. Use port 8043, to connect to the printer over HTTPS.

Refer to the printer documentation for specifics regarding installation and set up of the printer driver and library, as well as use of the printer’s API library.

Note: The SDK, driver and documentation for the sample Epson TM-H6000IV-DT Series printer can be found here.

Note the file name and path of the Javascript printer driver. You will need to reference this in the UI maps that generate the print dialog box from the payment transaction portals (see **Configuring and Updating UI Maps and BPA Scripts** on page 6-3.

Oracle Utilities Customer Cloud Service Configuration

Configuration of Oracle Utilities Customer Cloud Service includes the following:

- Configuring the Point of Sale Printer Integration Master Configuration
- Configuring and Updating UI Maps and BPA Scripts
- Configuring and Updating Tender Sources

Configuring the Point of Sale Printer Integration Master Configuration

To enable printing from the three payment transactions, you must define the BPA script used to launch the print dialog box from each type of transaction in the Point of Sale (POS) Printer Integration (C1-PointOfSaleIntegConfig) master configuration. The base product provides three processing types (one for each payment transaction that supports printing) and corresponding sample BPA scripts:

Processing Type	Sample BPA Script
POS Printing - Payment Event	Payment Event Add Print (C1-PEAddPrt)
POS Printing - Payment Event Quick Add	Payment Event Quick Add Print (C1-PEQAddPrt)
POS Printing - Payment Quick Add	Payment Quick Add Print (C1-PyQAddPrt)

Define a BPA script for each of the processing types you want to support.

Configuring and Updating UI Maps and BPA Scripts

The BPA scripts referenced on the Master Configuration each reference a UI map that's used to define the print dialog box. The base product provides sample UI maps for each of the sample BPA scripts listed above:

Sample BPA Script	Sample UI Map
Payment Event Add Print (C1-PEAddPrt)	Payment Event Print Control (C1-PayEventAddPrint)
Payment Event Quick Add Print (C1-PEQAddPrt)	Payment Event Quick Add Print (C1-PayEventQuickAddPrint)
Payment Quick Add Print (C1-PyQAddPrt)	Payment Quick Add Print (C1-PaymentQuickAddPrint)

Since the sample UI maps contain printer-specific information, the UI maps and the referencing BPA scripts must be copied and configured accordingly.

The UI maps must be updated to include specific information about the printer being used, including:

- Javascript Printer Library
- Printer Actions and Text Composition and Formatting

Javascript Printer Library

Each UI map must be updated to reference the file name and path of the Javascript printer library installed on the printer server (see **Printer Installation** on page 6-2).

Update the appropriate UI Map for each of the processing types you want to support. The printer library should be defined in a <script> element in the <head> element of the UI map's schema. Replace the ePOS-Javascript-SDK-path placeholder text in the UI map with the path of the ePOS.

Example: The sample below shows the location of the ePOS path placeholder in the UI map code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/TR/xhtml1/strict">
    <head>
        <title oraMdLabel="TITLE_CIPPEQPCTL"></title>
        <link href="cisDisabled.css" type="text/css"
rel="stylesheet" />
        <script type="text/javascript" src="c1/ccbUtil/
c1Utils.js"></script>
        <script type="text/javascript" src="ePOS-Javascript-SDK-
path"></script>
        <oraInclude map="C1-POSPayAmountFormatterJS"/>
        <script type="text/javascript">
```

Printer Actions and Text Composition and Formatting

Each UI map must also be updated to include the specific printer actions and appropriate text composition and formatting for each type of receipt you wish to print.

Use the printer library API to generate code for the printer actions and formatting you wish to use, and include that code in the UI map. For example, the Payment Event Quick Add Print (C1-PayEventQuickAddPrint) UI map includes the following functions for connecting to the printer, composing receipts, and printing receipts:

```
function printReceipt() {
// Connect to printer
    isPrintReceipt = true;
    connect();}

function connect() {
    // Get IP address from Tender Source
    id('tenderControlId').value = myPageData.TNDR_CTL_ID;
    oraInvokeSS('C1-GetPrntIP','getPrinterIP');
    var tenderSource = id('tenderSource').value;
    ipAddress = id('printerIPAddress').value;
    port = id('printerPort').value;
    if (ipAddress == '' || port == '') {
        // ERROR: IP and/or port missing
        main.showErrorMessage(11111, 80701, null, null, window,
[myPageData.TNDR_CTL_ID,tenderSource]);
        return;
    }
    ePosDev.connect(ipAddress, port, callback_connect);
}

function callback_connect(resultConnect) {
    if ((resultConnect == 'OK') || (resultConnect ==
'SSL_CONNECT_OK')) {
        if (isPrintReceipt) {
            ePosDev.createDevice('local_printer',
ePosDev.DEVICE_TYPE_PRINTER, {'crypto' : true, 'buffer' : false},
callback_createDevice);
        } else {
            ePosDev.createDevice('local_printer',
ePosDev.DEVICE_TYPE_HYBRID_PRINTER, {'crypto' : true, 'buffer' :
false}, callback_createDevice);
        }
    } else {
        // ERROR: Connection to the printer can not be made
        main.showErrorMessage(11111, 80702, null, null, window,
[resultConnect]);
    }
}

function callback_createDevice(deviceObj, resultCreateDevice) {
    retry++;
    if (resultCreateDevice == 'OK') {
        printer = deviceObj;
        retry = 0;
        if (isPrintReceipt) {
            composeReceipt();
            printer.addFeedLine(8);
            printer.send();
        } else {

printer.EndorsePrinter.addTextFont(printer.EndorsePrinter.FONT_B);
            composeEndorsement();
            printer.EndorsePrinter.send();
        }
        disconnectPrinter();
    } else if (resultCreateDevice == "DEVICE_IN_USE" ){
        if (retry < 5 ) {
            setTimeout(function () {
                ePosDev.createDevice('local_printer', 
ePosDev.DEVICE_TYPE_PRINTER, {'crypto' : true, 'buffer' : false}, 
callback_createDevice);
                }, 3000);
        }
    } else {
       // ERROR: Connection to the printer can not be made.  Create 
device error.
        main.showErrorMessage(11111, 80703, null, null, window, 
[resultCreateDevice]);
    }
}

function composeReceipt() {
    printer.addTextAlign(printer.ALIGN_CENTER);
    printer.addText(myPageData.USER_ID);
    printer.addText(' ');
    printer.addText(myPageData.TNDR_CTL_ID);
    printer.addText(' ');
    printer.addText(main.convertInternalDateToLo
    cal(myPageData.PAY_DT));
    printer.addText('\n\r\n');
    var receiptList = main.model.getList('PRINT');
    // Set max width of the printer output
    setMaxWidth(isPrintReceipt);
    // Determine maximum payment amount length. This is used for 
formatting the payment amount
    setMaxPayAmountLength(receiptList,myPageData.CURRENCY_CD);
    for (var x = 0; x < receiptList.elements.length; x++) {
        var elem = receiptList.elements[x];
        if (elem.PAY_EVENT_ID != '' || elem.ACCT_ID != '' ||
            elem.PAY_AMT != 0 || elem.TENDER_TYPE_CD != '') {
            printer.addTextAlign(printer.ALIGN_LEFT);
            printer.addText(elem.ACCT_ID);
            printer.addText(' ');
            printer.addText(elem.ACCT_CHECK_DIGIT);
            printer.addText('\n');
            var tndrDesc = '';
            var parameters = {
                FK_REF_CD: 'C1-TNDTY',
                FK_VALUE1: elem.TENDER_TYPE_CD
            };
            var result = oraGetFkRefInfoFromServer(parameters);
           if (result != null && result != '' && result.INFO_DESCR 
!= null) {
                tndrDesc = '  ' + result.INFO_DESCR;
            }
           // Determine if payment amount will be on the same line 
as tender type
            var isWrap = shouldWrap(tndrDesc);
            printer.addText(tndrDesc);
            if (isWrap) {
                printer.addText('\n');
            }
            var payAmt = 
main.convertInternalMoneyToLocal(elem.PAY_AMT, 
myPageData.CURRENCY_CD);
          // Pad the payment amount with the appropriate spaces to 
ensure that amounts are right-aligned
```

```
            printer.addText(getPayAmountPadding(tndrDesc, payAmt,
isWrap) + payAmt);
            printer.addText('\n');
        }

}
}
```

Refer to the printer documentation for specifics regarding use of the printer's API library.

> **Note**: The SDK, driver and documentation for the sample Epson TM-H6000IV-DT Series printer can be found here.

Refer to the Payment Event Quick Add Print (C1-PayEventQuickAddPrint) UI map for additional sample functions and text composition examples.

Configuring and Updating Tender Sources

You must also update the Tender Sources used for the payment transactions you need to support.

The functions defined in the UI maps access the printer library (installed on the printer server, see **Printer Installation** on page 6-2) via the printer's IP address and port number.

> **Note**: The UI map derives the printer IP address and port from the Tender Source using the Get Printer IP Address and Port (C1-GetPrntIP) service script.

The printer's IP address and port number should be specified in the **Printer ID Address** and **Port Number** fields on the Tender Sources used by the Tender Controls of the payment transactions for which printing is supported.

> **Note**: The default port can vary by printer model. For instance, the default port for an Epson TM-H6000IV-DT is 8008.

Refer to the *Business User Guide* for more information about Tender Sources.