## What is proration

*Adjusting a customer's bill amount to reflect any plan changes made in the middle of a billing cycle.*

*Proration with Upgrades*

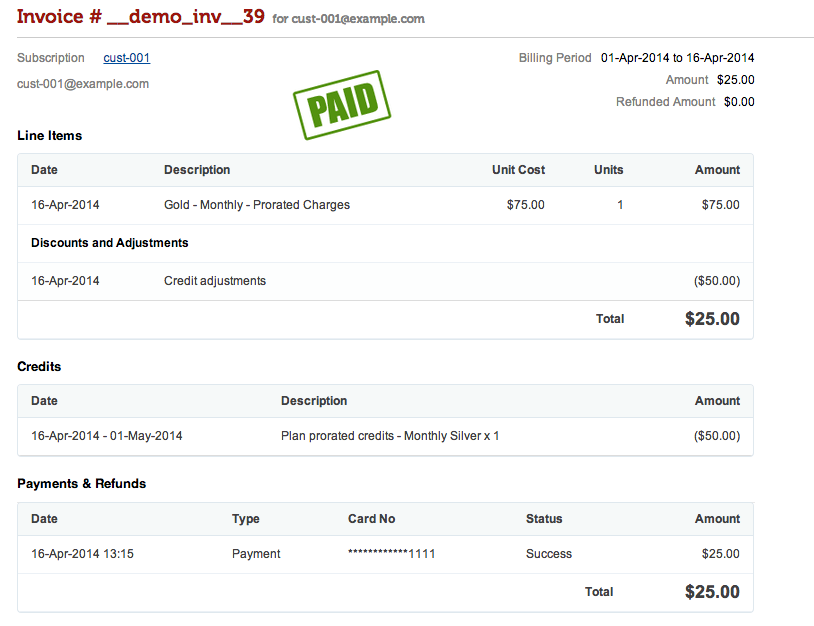
When a subscription is upgraded, calculate the amount for the remaining period of the new plan and add that as a line item to the invoice generated during the plan change.

Current Plan: $100 / month

New plan: $150 / month

Proration logic:

* $100 paid at the beginning of the billing cycle.
* Assume, changes are made exactly in the middle of the billing cycle.
* Prorated “consumed” charge $50.
* Prorated credit remaining is $50.
* Additional charge on new plan for the remaining period: $75
* Net additional charges after adjusting credits: $25



*Proration with Downgrades*

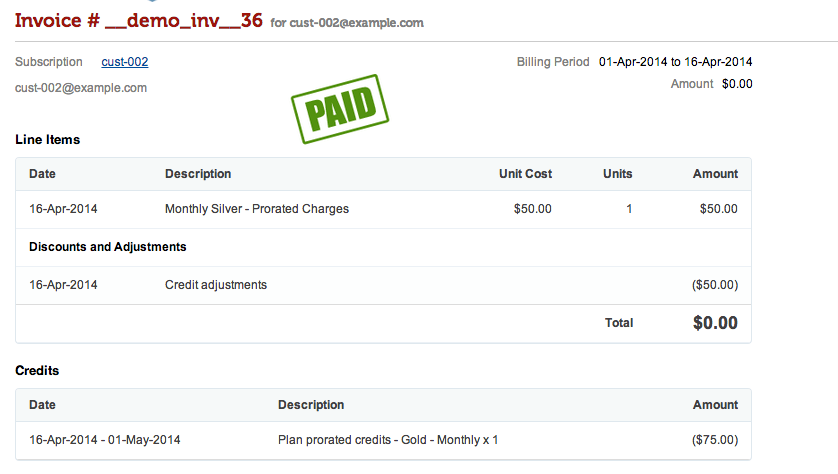
When a subscription is downgraded, the customer is more likely to have paid more than the expected usage in the middle of the billing cycle. There will be a net credit to the customer. This needs to reflect unambiguously in the invoice line items, with appropriate charges.

Current Plan: $150 / month

New Plan: $100 / month

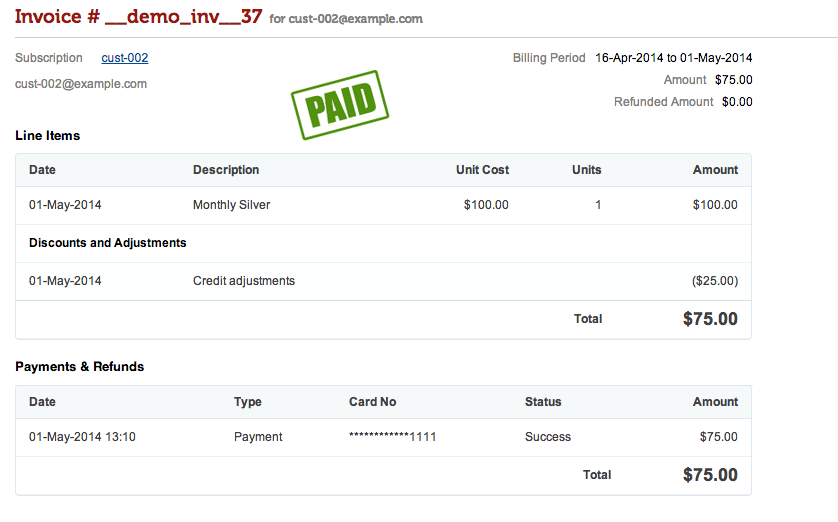
Proration Logic:

* $150 paid at the beginning of the billing cycle
* Assume, changes are made exactly in the middle of the billing cycle.
* Prorated “consumed” charge $75
* Prorated credit remaining is $75
* The charge on the new plan for the remaining period: $50



Net additional credits after adjusting credits: $25.

Here is what the next bill would look like:



## Proration Credit

https://knowledgecenter.zuora.com/Billing/Subscriptions/Subscriptions/G\_Proration/B\_Proration\_Credit

There are situations where a credit needs to be issued back to a customer who has paid for a service period in advance. For instance, if a customer cancels a subscription early or removes a product from their subscription, you can choose to issue a credit for the unused service period.

For example, imagine a scenario where:

* the bill cycle day of the subscription rate plan charge is set as *Default from Customer Account*
* the bill cycle day of the account is set as **1st of the month**
* the subscription billing trigger date is **10/1/2014**
* the billing period of the rate plan is set to **quarterly**
* the rate plan is set to align to subscription or align to term such that the first full quarter period is 10/1/2014 to 12/31/2014
* the subscription is **canceled early on 10/15/2014**

The amount of prorated credit issued depend on the combination of the **Prorate recurring charges for partial period?**and **Bill recurring charges for partial month (with monthly based billing periods)?**settings, as specified in **Billing >**[**Define Billing Rules**](https://knowledgecenter.zuora.com/Billing/Billing_and_Payments/Billing_Settings/Define_Billing_Rules)**.** In all cases the first invoice charges for the full quarterly service period from the subscription trigger date to the bill cycle day, 10/1/2014 – 12/31/2014.

For charges involving in proration credit, the charge names displayed as line items on invoices vary in different situations:

* If a charge is billed for a partial month or period, the charge name ends with **Proration**.
* If a charge is partially credited back, the charge name ends with **Proration Credit**.
* If a charge is fully credited back, the charge name ends with **Credit**.

## Provisioning queue

is **a way to handle a considerable number of order items**. A full list of order items (all items of all orders) can always be filtered by many parameters, such as order type and status, order component type and status, service plan, service template, and the like.

**cancel\_unapplied\_service\_credits\_m**

This API cancels all the unapplied service credits and any partially used recurring credits on an account or master plan instance.

What is edgecase scenario??

https://www.applause.com/blog/how-to-find-test-edge-cases

An edge case is **a problem or situation that occurs only at an extreme (maximum or minimum) operating parameter**. For example, a stereo speaker might noticeably distort audio when played at maximum volume, even in the absence of any other extreme setting or condition. An edge case can be expected or unexpected.

### Common Edge Cases

This is a bit of an oxymoron, as the whole point of an “edge case” is that it’s not happening affecting many users. But there are still certain edge cases that can be fairly common among different apps or websites overall. Some of these include:

* Crashes. One of the most common edge cases is when an app crashes without an obvious or very reproducible path. Oftentimes, the developer would be able to look at the code and determine the cause. But as a tester, you may not be able to easily reproduce the crash. (Good luck getting a screen recording!)
* A section of the app or website taking an extra 5 seconds to load once in awhile (regardless of network speed).
* Audio continuing to play in the background after closing a video screen on occasion.

Some edge cases are unique to the app or website they occur on. But others can be seen on many different types of websites or mobile apps.

Groovy script to create a csv file and save the data in the csv file

import java.util.Calendar;

import java.util.Date;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import com.eviware.soapui.support.XmlHolder;

import org.apache.commons.lang.StringUtils;

import com.jcraft.jsch.\*;

public String getJulianDateForCurrentDate(){

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyyMMdd");

Date date = dateFormat.parse(dateFormat.format(new Date()));

StringBuilder sb = new StringBuilder();

Calendar cal = Calendar.getInstance();

cal.setTime(date);

return sb.append(cal.get(Calendar.YEAR)% 100).append(String.format("%03d", cal.get(Calendar.DAY\_OF\_YEAR))).toString();

}

public String getCurrentDateForCurrentDate(){

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyyMMdd");

String date = dateFormat.format(new Date());

return date;

}

public String getCurrentTimeForCurrentDate(){

SimpleDateFormat dateFormat = new SimpleDateFormat("HHmmSS");

String date = dateFormat.format(new Date());

return date;

}

public void createInputFileDirectory(){

// CREATE INPUT FILES FOLDER IF DOES NOT EXISTS

String ProjectPath = context.expand('${projectDir}')

File inputFileDirectory = new File(ProjectPath + "/" + context.expand('${#TestSuite#InputFileFolder\_Local}'));

if (! inputFileDirectory.exists()){

inputFileDirectory.mkdir();

log.info("Input file Directory Created..!!!");

}else{

log.info("Input file Directory already Exists..!!!");

}

}

public void generateInputFile(){

def groovyUtils = new com.eviware.soapui.support.GroovyUtils(context)

BufferedWriter myWriter;

StringBuilder builder;

createInputFileDirectory();

String ProjectPath = context.expand('${projectDir}')

def outputFilePath = ProjectPath + "/" + context.expand('${#TestSuite#InputFileFolder\_Local}') + "/chronicle-"+context.expand('${#Project#CTSCustomerNotification\_Client\_No}')+"-"+getCurrentDateForCurrentDate()+"-"+getCurrentTimeForCurrentDate()+".csv";

log.info("Input File Path : " + outputFilePath)

testRunner.testCase.testSuite.setPropertyValue("InputFilePath\_Local", outputFilePath)

myWriter = new BufferedWriter(new FileWriter(outputFilePath, true))

builder = new StringBuilder()

def header = "";

def newLineText = "";

def BU;

def customerID;

def customerName;

def ariaAccountNumber;

def creditAmount;

def ariaInvoiceNumber;

def businessUnit;

def documentDate;

def currency;

def customerPONumber;

def dueDate;

def charges;

def chargesNotYetDue;

for(int i=1; i<=2; i++){

//testRunner.runTestStepByName("Delay")

testRunner.runTestStepByName("DS-Account")

testRunner.runTestStepByName("DS-ChronicleFile")

testRunner.runTestStepByName("create\_acct\_complete\_m")

testRunner.runTestStepByName("get\_acct\_supp\_fields\_m")

header = "Customer ID"+","+ "Customer Name"+","+ "Aria Account Number"+","+ "Business Unit"+","+ "Customer PO Number"+","+ "Aria Invoice Number"+","+ "Document Date"+","+ "Billing Period"+","+ "Due Date"+","+ "Currency"+","+ "Charges Not Yet Due"+","+ "0-30 Days" +","+"31-60 Days"+","+"61-90 Days"+","+"91-120 Days"+","+"121-150 Days"+","+"151-180 Days"+","+"181+ Days"

if(i==1){

myWriter.write(header)

}

BU = context.expand( '${DS-ChronicleFile#BusinessUnit}' );

customerID = context.expand( '${create\_acct\_complete\_m#Response#//\*:client\_acct\_id[1]/text()}' )

customerName = BU+"&T (Tech) Inc LLC."

ariaAccountNumber=context.expand('${create\_acct\_complete\_m#Response#//\*:out\_acct[1]/\*:acct\_no[1]/text()}' )

businessUnit = BU;

currency = context.expand( '${DS-ChronicleFile#Currency}' )

customerPONumber=context.expand('${get\_acct\_supp\_fields\_m#Response#//\*:supp\_field[3]/\*:supp\_field\_value[1]/text()}' )

for(int invoiceCount=1;invoiceCount<=3;invoiceCount++){

testRunner.runTestStepByName("gen\_invoice\_m")

ariaInvoiceNumber = context.expand( '${gen\_invoice\_m#Response#//\*:invoice\_no[1]/text()}')

documentDate=context.expand('${gen\_invoice\_m#Response#//\*:invoice\_items[1]/\*:start\_date\_range[1]/text()}')

dueDate=context.expand('${gen\_invoice\_m#Response#//\*:invoice\_items[1]/\*:end\_date\_range[1]/text()}')

charges=context.expand('${gen\_invoice\_m#Response#//\*:out\_invoices\_list[1]/\*:invoice\_charges\_after\_tax[1]/text()}')

charges = (Double.parseDouble(charges)/8).toString();

newLineText=customerID+","+customerName+","+ariaAccountNumber+","+businessUnit+","+customerPONumber+","+ariaInvoiceNumber+","+documentDate+","+"-"+","+dueDate+","+currency+","+charges+","+charges+","+charges+","+charges+","+charges+","+charges+","+charges+","+charges

builder.append("\n")

builder.append(newLineText)

myWriter.newLine()

myWriter.write(newLineText)

// Write credits to file

testRunner.runTestStepByName("apply\_cash\_credit\_m")

creditAmount = context.expand('${apply\_cash\_credit\_m#Response#//\*:amount[1]/text()}')

creditAmount = (Double.parseDouble(creditAmount)/8).toString();

newLineText=customerID+","+customerName+","+ariaAccountNumber+","+businessUnit+","+customerPONumber+","+""+","+""+","+"-"+","+dueDate+","+currency+","+"-"+creditAmount+","+"-"+creditAmount+","+"-"+creditAmount+","+"-"+creditAmount+","+"-"+creditAmount+","+"-"+creditAmount+","+"-"+creditAmount+","+"-"+creditAmount

builder.append("\n")

builder.append(newLineText)

myWriter.newLine()

myWriter.write(newLineText)

}

}

log.info("File generated Successfully !!!");

myWriter.flush();

myWriter.close();

}

public void uploadFileToFTP(){

String ProjectPath = context.expand('${projectDir}')

String sftpHost = context.expand('${#TestSuite#SFTPHOST}');

int sftpPort = Integer.parseInt(context.expand('${#TestSuite#SFTPPORT}'));

String sftpUserName = context.expand('${#TestSuite#SFTPUSER}');

String sftpPassword = context.expand('${#TestSuite#SFTPPASS}');

String sftpRemoteDirectory = context.expand('${#TestSuite#SFTPWORKINGDIR}');

String fileName;

Session session = null;

Channel channel = null;

ChannelSftp channelSftp = null;

System.out.println("preparing the host information for sftp.");

try {

JSch jsch = new JSch();

session = jsch.getSession(sftpUserName, sftpHost, sftpPort);

session.setPassword(sftpPassword);

java.util.Properties config = new java.util.Properties();

config.put("StrictHostKeyChecking", "no");

session.setConfig(config);

session.connect();

System.out.println("SFTP Host connected.");

log.info("Connected")

channel = session.openChannel("sftp");

channel.connect();

System.out.println("SFTP Channel Opened and Connected.");

channelSftp = (ChannelSftp) channel;

channelSftp.cd(sftpRemoteDirectory);

fileName = context.expand('${#TestSuite#InputFilePath\_Local}');

//log.info fileName

File f = new File(fileName);

channelSftp.put(new FileInputStream(f), f.getName());

log.info("File transfered successfully to host.");

} catch (Exception ex) {

log.info("Exception Occurred !!!");

} finally {

channelSftp.exit();

channel.disconnect();

session.disconnect();

}

}

// GENENRATE INPUT FILE

generateInputFile();

//Upload file to SFTP

//uploadFileToFTP();

Groovy script for date generation

import java.util.Calendar;

import java.util.Date;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.time.LocalDate;

import java.time.format.DateTimeFormatter;

def virtualDateString = context.expand('${get\_virtual\_datetime\_m#Response#//\*:virtual\_date[1]/text()}')

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");

LocalDate virtualDate = LocalDate.parse(virtualDateString, formatter);

LocalDate effectiveDate = virtualDate.plusDays(1);

log.info("effectiveDate : " + effectiveDate)

testRunner.testCase.setPropertyValue("effectiveDate", effectiveDate.toString())