**Group 2: Project Design**

**Current Document Revision: 3**

**Currency Exchange Program – Exchange!**

**Reginald Carey, Jason Dudash, Robert Kibler, Brandon Trexler**

**University of Maryland: University College**

**CMSC 495 (7980)**

**April 10, 2016**

**Revision History Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision Number** | **Date** | **Description** | **Name** |
| **1** | **Apr 9, 2016** | **Initial Document** | **Robert Kibler** |
| **2** | **Apr 10, 2016** | **Add Revision History**  **Add Startup Diagrams**  **Add Class Diagrams**  **Update Event Diagrams** | **Reginald Carey** |
| **3** | **Apr 10, 2016** | **Added pseudo code** | **Reginald Carey** |

# Class Diagram

Macintosh HD:Users:ReginaldCarey:Documents:UMUC:SPRING 2016:CMSC 495 Current Trends and Projects in Computer Science:CMSC495:Class Diagram.png

# Event Trace Diagrams

The following event trace diagrams cover the initialization of the application and the two use cases of the application. They describe the pre-conditions and post-conditions and there is a description

# Startup Diagrams: Initialization of UI models

## Description:

User starts up the application. As part of the initialization prior to the UI being presented, the list of currency codes is needed so the user may make a selection from one of them.

## Pre-condition:

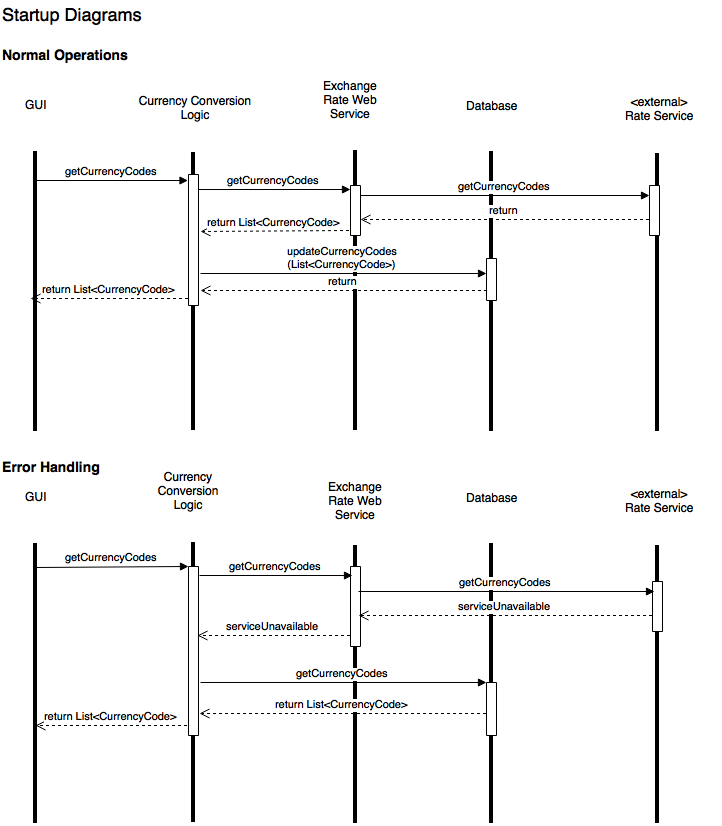
Internet access is available.

## Post-condition:

The database has been updated or added to with the set of legal currency codes. The UI models for lists of currency codes have been initialized. User interface rendered to the screen and is ready for input.

## Error-condition:

The connection to the internet rate service is not available for what ever reason. The UI models for lists of currency codes have been initialized with cached data in the database. User interface rendered to the screen and is ready for input.



# Scenario 1: User selects currencies

## Description:

User selects a currency from each dropdown box but has not yet entered anything into the input field. The GUI must detect this condition by observing the currently selected source and target currencies. Once both currencies are selected Scenario 1 ensues.

## Pre-condition:

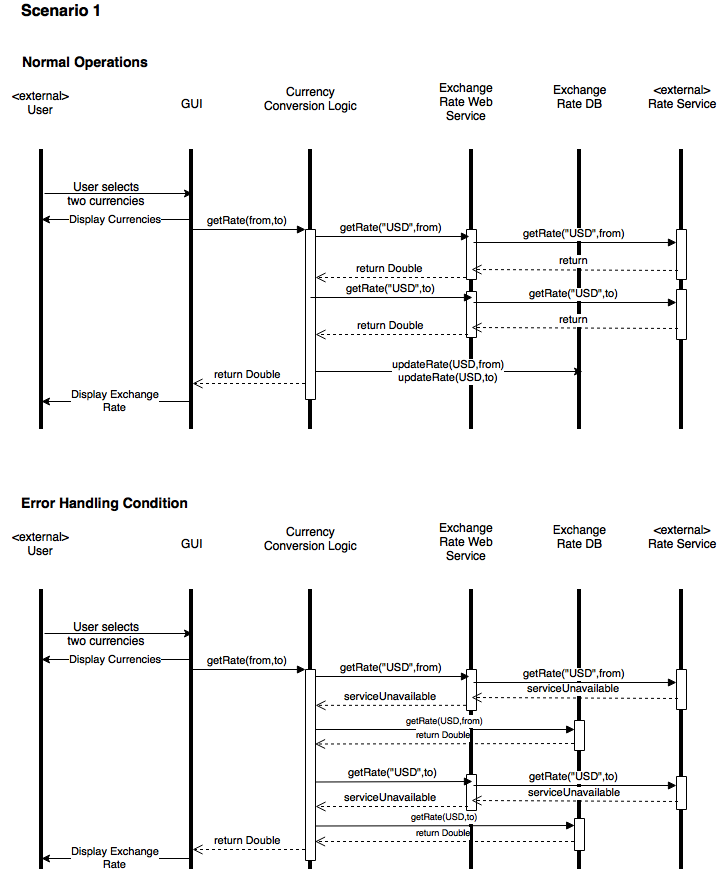
The application has been started and is running and the user interface is displayed. The web rate service is available. The Startup initialization has completed.

## Post-condition:

The exchange rate between the two selected currencies is displayed to the user. The exchange rate is updated to the offline database.

## Error-condition

The error condition occurs when the web service is unavailable. In this mode, the system reverts to using the database for the rate information.



# Scenario 2: User selects currencies and inputs value

## Description:

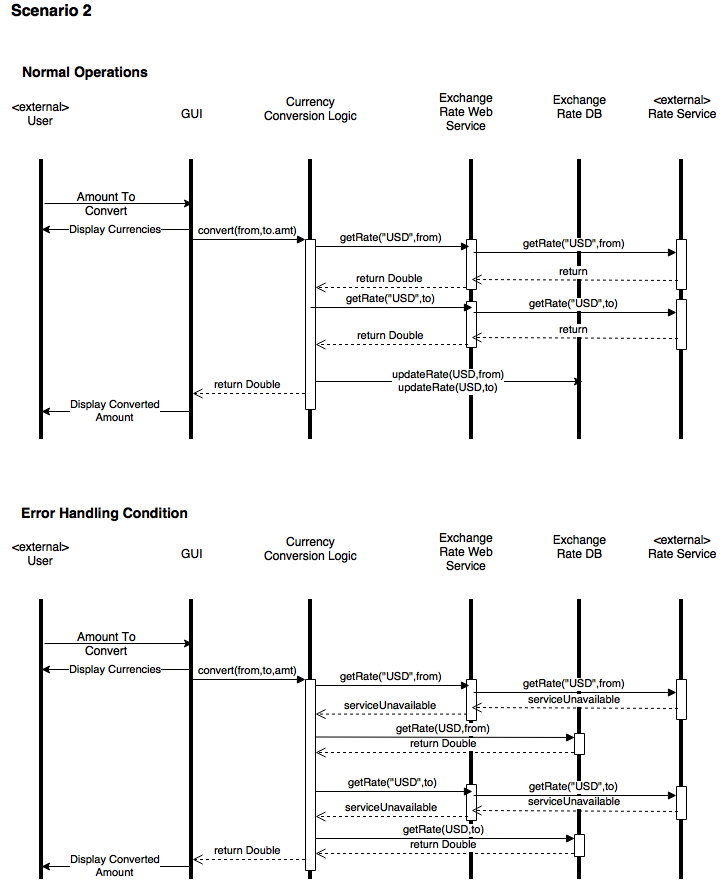
User selects a currency from each dropdown box and then enters an amount into the base currency input field.

## Pre-condition:

The application has been started and is running and the user interface is displayed. The web rate service is available.

## Post-condition:

The exchange rate between the two selected currencies is displayed to the user. The Exchange rate is updated to the offline database. The exchange rate that is displayed is used to calculate the value of the base currency amount in the target currency.



# Pseudo Code

## ExchangeRateDB

conection getConnection (

try {

Open Database connection

}

return connection;

)

updateRates {

Connection conn getConnection;

SQL drop table Rates

SQL create table Rates{

pk currencyCode

col conversionRate

col time

}

SQL INSERT INTO RATES {

parse through webservice.getAllRates(List) and insert into database

}

}

getCurrencyCodes() List:CurrencyCodes {

Connection conn getConnection;

List<CurrencyCodes> = new List;

conn.preparedStatement(Select CurrencyCode FROM Rates) {

add each parsed database code into List<CurrencyCodes>

}

Return List<CurrencyCodes>;

}

getRate(CurrencyCode) Double {

Connection conn getConnection;

Double currentRate = null;

SELECT conversion FROM Rates WHERE this.CurrencyCode = CurrencyCode {

currentRate = results.getDouble("conversionRate");

}

}

getUpdateTime(CurrenyCode) String {

Connection conn getConnection;

String updateTime = null;

SELECT time FROM rates WHERE this.CurrencyCode = currencyCode{

updateTime = results.getString("time");

}

}

## CurrencyConversionLogic

Class CurrencyConversionLogic{

double conversionRate

list currencycodes

double getRate(from, to)

{

this.conversionRate = dbService.getRate(from) / dbService.getRate(to)

}

double convert(from, to, amount){

Call this.getRate(from, to)

return amount \* conversionRate

}

}

## ExchangeRateWebService

Class ExchangeRateWebService

string webURL

list getCurrencyCodes()

{

try

{

get currencyRateXML from webURL for all currencies

}

catch (connection error, web service error, file error)

{

unable to update db

exit

}

stream currencyRateXML

codelist = SAXParse xml for all currency codes;

return codeList

}

list getSingleRate(currencyCode)

{

try

{

get currencyRateXML from web for single code

}

catch (connection error, webURL service error, file error)

{

unable to access webservice

exit

}

SAXparse xml for rate

return rate

}

list getAllRates()

{

try

{

get currencyRateXML from web for all currencies

}

catch (connection error, webURL service error, file error)

{

unable to access webservice

exit

}

SAXparse xml for rateList

return rateList

}

}

## CurrencyCode

Public Class CurrencyCode {

private final String code

private final String name

private Boolean smallestDenomination

public CurrencyCode(String code, String name, Boolean sDen) {

this.code = code

this.name = name

this.smallestDenomination = sDen

}

public String getCode() {

return code

}

public String getCurrencyName() {

return name

}

public Boolean isSmallestDenomination() {

return smallestDenomination

}

}

## GUI

# Unresolved Risks