ACTUS Risk Factor Interfaces and Deposit Account Modelling: an introduction and user guide

Risk Modelling with ACTUS Contracts: a user handbook

Francis N Parr

January/February 2023

(released with the approval of the ACTUS Architecture and API's Working Group*** – to be confirmed)

1 Introduction

1.1 Purpose of this document

This document is intended to provide ACTUS users with:

- Background information on ACTUS treatment of Risk Factors in the actus -webapp application
- A User guide and documentation of user interfaces for working with ACTUS contracts and risk Factor models – *the current proposal of the ACTUS WG on Architecture and APIs*
    - A RiskFactor enabled API enabling definition and use of risk scenarios and simple risk factor models for (1) Market Risk (2) Prepayment risk, (3) Credit Loss risk and (4) Demand deposit risk
    - A Risk Factor SPI allowing Java developers to integrate arbitrary external risk models with the ACTUS core cash flow generation library
- More detailed information and examples on the specific use of the risk Factor API to model demand deposit accounts using the risk factor API above
- An Appendix on where to find the (open source) code of the version of actus-webapp which supports this proposed Risk Factor enable API
- Design overview of extensions to actus-webapp for this Risk Factor API

1.2 Outline of Document

- 2. ACTUS context
    - ACTUS background: contract terms and cashflow dependency on risk factors
    - Overview of current actus-webapp and its support of the ACTUS demo

- 3. Market risk factors using V1.0 webapp/eventBatch interface
    - Market risk factor data and variable rate contracts
    - Actus-webapp Version1.0 (Current) entry points
    - Example variable rate contract simulation using webapp/eventsBatch
- 4. Motivation / requirements for additional risk Factor capabilities in actus-webapp
    - non market factor risk models – prepayment, credit default, exercise
    - Keeping risk modelling "external" to the Actus core
    - The "risk observer concept"
    - A risk factor store – scenario based risk factor simulation
- 5. Proposed Risk Factor API enabling user specification of simple risk models
    - Create scenario, create risk factor ,
    - The API – SPI architecture -
- 6. Proposed webapp V1.1 * –
    - Reusing the FDIC-webapp + requirements get working,
    - Adding DepositTransaction risk model
    - Open webappV1.1 will be the SPI "Documentation"
    - Categories of ACTUS user
- 7. An example – using webpp1.1* model Demand deposit risk on UMP "accounts"
    - Characteristics of UMPS and Demand Deposit accounts
    - Defining the DepositTrx risk model
    - Creating a Deposits Scenario
    - Simulating a deposit account UMP contract
- 8. The Future - do portfolios and aggregation/need something similar to Risk ?
- Appendixes:
    - (1) building stopping and starting webapp servers
    - (2) code repositories
    - (3) documentation resources
    - (4) the DaDFir3 demo – other resources

2 ACTUS background – contracts, cash flows, risk factor dependencies

2.1 ACTUS as a standard for contract terms – relation to risk

- ACTUS (Algorithmic Contract Terms Universal Specification) allows most financial contracts to be defined using terms from which the cash flows of the contract can be deterministically and algorithmically generated for any given *risk scenario*.
- A risk scenario is a set of risk factor models specifying how to treat any uncertainty which will affect a contract's future cashflow behavior.
- Examples of risk factor models include (1) market models projecting expecting future behavior of interest rates and instrument prices (2) models for external risk such as default risk, prepayment risk , behavior of user deposit accounts etc.
- Different users of ACTUS will have different perspectives on which risk factors are important to them and on the type and complexity of risk factor models which they

want to use with the actus-core cash flow generation library and portfolios of ACTUS defined contracts.

- The variety and "subjectivity" of risk models contracted with the deterministic actus-core logic generating cash flows for each contract type requires that risk modelling be kept "external" to the actus-core library interacting with it through narrow well defined interfaces (SPIs)

2.2 The ACTUS demo implemented in current (v1.0) actus-webapp

- Potential ACTUS users are encourage to familiarize themselves with ACTUS contracts and view their cashflow behavior by experimenting with the ACTUS "browser" demo available at https://demo.actusfrf.org/demo and publicized on the ACTUS web site at https://actusfrf.org.
- This demo is helpful is making it easy for a user to view the terms of simple sample ACTUS contract for each of the supported contract types, and to generate graphical and text descriptions of the cashflows for any selected sample. The user is also free to modify the terms and observe the resulting changes in contract cashflow.
- The ACTUS demo is implemented as a web service built from the actus-webapp application which in turn calls the actus-core cashflow generation library.
- There is published guidance on the ACTUS website *** on how to build an ACTUS demo server using open source code from https://github.com/actusfrf/actus-webapp and explaining how to link this to an actus-core.jar built from the open source code in https://github.com/actusfrf/actus-core. The actus-core library provides the actual cashflow generation logic for any contract and is invoked by an actus-webapp web server (implemented with Apache Tomcat **) listening for cashflow generation requests.

2.3 Version 1.0 actus-webapp support for variable rate contracts

- A shortcoming of the current actus-webapp *demo* is that there is no support for variable rate instruments. Terms which refer to underlying reference data and spreads can be entered BUT there is no facility in the browser demo interface to enter market risk factor data, hence no way to use the browser demo to see, cashflows for contracts whose interest rate or value depends on some underlying reference market rate.
- Separate from the ACTUS browser demo (with its selectable sample contracts and graphical rendering of generated cash flows), current actus-webapp provides additional entry points which can be accessed through curl commands and JSON parameter lists. These entry points allow "inline" market risk factor data to be entered with ACTUS contract specifications and a cash flow generation request. The generated cashflow event sequences are returned as a JSON output string.

- The actus-webapp 1.0 entry points are:

Entry point	*Description*
/demo	Browser access to the ACTUS demo
/demo-meta	JSON commands to manage demo sample data etc
/events	JSON request to simulate a contract with in-line market risk data; returns JSON data structure with a sequence of generated cash flow events or a brief error message
/eventsBatch	JSON request to simulate a batch of contracts with shared in-line market risk data; returns a JSON data structure with a list of generated cash flow event sequences or error messages

- The webapp/events and webapp/eventsBatch entry points can generate cash flows for *variable rate* contracts by accepting market risk factor data in-line with the contract simulation request and passing this data as an input to the actus-core library cash flow generation.

3 Using V1.0 actus-webapp/eventsBatch to generate cash flows

- We will focus on describing use of the /eventsBatch entry point to actus-webapp since it is a generalization of the /events entry - allowing multiple contracts to be sent for simulation and cash flow generation in a single request. When performing analytics on a portfolio of contracts, which typically use a common set of underlying market reference data, there can be efficiency gains from batching the contracts.
- We will describe an example which is a variable rate contract with cash flows which are sensitive to an underlying market risk factor reference index. The /eventsBatch entry point can also be used to request simulation of fixed rate contracts with no market risk factor dependencies.
- Curl commands are the usual access method to submit a request to a non-rendering (non-browser) web service.
- Since there is no formal documentation of the v1.0 actus-webapp/eventsBatch interface publicized on the ACTUSfrf website, we will use this document to illustrate curl access to /eventsBatch with an example described in some detail.
- The public ACTUSfrf demo dervice (running version 1.0 actus-webapp) is installed at https://demo.actusfrf.org:8080 hence url https://demo.actusfrf.org:8080/eventsBatch is a request to the /eventsBatch entry point on that server
- The format of a curl request to that /eventsBatch entry point is:
  - `> curl -v -H 'Content-Type: application/json' -data '<request>' https://demo.actusfrf.org:8080/eventsBatch`

- Where <request> is the JSON string specifying the contract and market data:

```
{"contracts":
  [{"calendar":"WEEKDAY",
    "businessDayConvention":"SCF",
    "contractType":"PAM",
    "statusDate":"2015-01-01T00:00:00",
    "contractRole":"RPA",
    "contractID":"Contract-01",
    "cycleAnchorDateOfInterestPayment":"2016-01-02T00:00:00",
    "cycleOfInterestPayment":"P1YL0",
    "nominalInterestRate":0.02,
    "dayCountConvention":"30E360",
    "currency":"USD",
    "contractDealDate":"2015-01-01T00:00:00",
    "initialExchangeDate":"2015-01-02T00:00:00",
    "maturityDate": "2020-01-02T00:00:00",
    "notionalPrincipal":1000,
    "premiumDiscountAtIED":0,
    "cycleAnchorDateOfRateReset": "2016-01-02T00:00:00",
    "cycleOfRateReset": "P2YL0",
    "rateSpread":0.01,
    "marketObjectCodeOfRateReset":"YC_EA_AAA"
   }],
"riskFactors":
  [{"marketObjectCode":"YC_EA_AAA",
    "base": 1.0,
    "data":
     [{"time":"2014-01-01T00:00:00",
       "value": 0.04
     ]}
  ]}
}
```

- We have “pretty printed” the <request> here – but it should be organized as a single record continuous JSON string when presented on the command line as illustrated in figure 1 below)

- The batch of contracts in this example case consists of a single PAM contract with contractID = “Contract-01”
- The terms of the contract are specified as a list of JSON name value pairs – with each name being an ACTUS contract term as defined in the ACTUS data dictionary
- This example PAM contract has a variable interest rate computed at contract Rate Reset times as a spread from the current value of market index “YC_EA+AAA”
- The in-line risk factor data consists of values for a single market reference index, “YC_EA_AAA” and in fact a single set of name value pairs specifying the value of that index at one point in time.
- The more usual case is that there is a time series of projected future and possibly historical values for each specified Market Object Code and there can be multiple Market Object Code time series specified in the risk Factor data of a single eventsBatch request

- The “base” parameter set here with value 1.0 for for MarketObjectCode “YC_EA_AAA” determines how the values in the riskFactor time series will be interpreted. In this example “base” = 1.0, and the riskFactor has “value”= 0.04 on date 2014-01-01. This will be interpreted in actus-core as an interest rate of 4% pa. The same effect can be achieved with “base” = 100 and “value” = 4.0 . Having a scaling factor included in the market Object Code specification simplifies combining data from different sources with different conventions in the same risk Factor specification
- When using the /eventsBatch entry point to request simulation of fixed rate contracts with no market risk factor dependencies, the “riskFactors” term must still be present but the list of riskFactors can be empty, i.e. “riskFactors”: [ ]

Raw command line JSON for the example /eventsBatch request

```
curl –v –H 'Content–Type: application/json'  ––data '{"contracts":
   [{"calendar":"WEEKDAY","businessDayConvention":"SCF","contractType":"PA
   M","statusDate":"2015–01–01T00:00:00","contractRole":"RPA",
   "contractID":"Contract–01", "cycleAnchorDateOfInterestPayment":"2016–
   01–02T00:00:00", "cycleOfInterestPayment":"P1YL0",
   "nominalInterestRate":0.02,"dayCountConvention":"30E360","currency":"US
   D","contractDealDate":"2015–01–01T00:00:00","initialExchangeDate":
   "2015–01–02T00:00:00","maturityDate":"2020–01–02T00:00:00",
   "notionalPrincipal":1000,"premiumDiscountAtIED":0,
   "cycleAnchorDateOfRateReset":"2016–01–02T00:00:00","cycleOfRateReset":
   "P2YL0","rateSpread":0.01, "marketObjectCodeOfRateReset":"YC_EA_AAA"}],
   "riskFactors":[{"marketObjectCode":"YC_EA_AAA","base": 1.0,
   "data":[{"time":"2014–01–01T00:00:00","value": 0.04}]}]}'
   localhost:8082/eventsBatch
```

Figure 1 – Actual command line string for a curl eventsBatch request for a single PAM contract with risk factor data for Market Object Code “YC_EA_AAA” and no preceding command prompt “>”

- The result returned from an eventsBatch request is a JSON string consisting of a list with an entry for each contract in the batch. Each entry will be either a JSON sequence of the cash flow events for that contract OR, a short message indicating that the simulation for that contract failed.
- The result data for an eventsBatch request comes back as a single record JSON data string. For readability we show the results returned for the curl example above simulating PAM contract with contractID = “contract-01” in a “pretty printed” format:

```
[{"contractId":"Contract–01",
  "status":"Success",
  "message":"",
```

```
"events":[{"type":"IED",
           "time":"2015-01-02T00:00",
           "payoff":-1000.0,
           "currency":"USD",
           "nominalValue":1000.0,
           "nominalRate":0.02,
           "nominalAccrued":0.0
            },
           {"type":"IP",
            "time":"2016-01-02T00:00",
            "payoff":20.0,
            "currency":"USD",
            "nominalValue":1000.0,
            "nominalRate":0.02,
            "nominalAccrued":0.0
            },
           {"type":"RR",
            "time":"2016-01-02T00:00",
            "payoff":0.0,
            "currency":"USD",
            "nominalValue":1000.0,
            "nominalRate":0.05,
            "nominalAccrued":0.0
            },
           {"type":"IP",
            "time":"2017-01-02T00:00",
            "payoff":50.0,
            "currency":"USD",
            "nominalValue":1000.0,
            "nominalRate":0.05,
            "nominalAccrued":0.0
            },
           {"type":"IP",
            "time":"2018-01-02T00:00",
            "payoff":50.0,
            "currency":"USD",
            "nominalValue":1000.0,
            "nominalRate":0.05,
            "nominalAccrued":0.0
            },
           {"type":"RR",
            "time":"2018-01-02T00:00",
            "payoff":0.0,
            "currency":"USD",
            "nominalValue":1000.0,
            "nominalRate":0.05,
            "nominalAccrued":0.0
            },
           {"type":"IP",
            "time":"2019-01-02T00:00",
            "payoff":50.0,
            "currency":"USD",
            "nominalValue":1000.0,
            "nominalRate":0.05,
            "nominalAccrued":0.0
```

```
                    },
                   {"type":"IP",
                    “time":"2020-01-02T00:00", "payoff":50.0,
                    "currency":"USD",
                    "nominalValue":1000.0,
                    "nominalRate":0.05,
                    "nominalAccrued":0.0
                    },
                   {"type":"MD",
                    "time":"2020-01-02T00:00",
                    "payoff":1000.0,
                    “currency":"USD",
                    "nominalValue":0.0,
                    "nominalRate":0.05,
                    "nominalAccrued":0.0
                    }
               ]}
          ]
```


-

Example of returned raw JSON data stream


```
[{"contractId":"Contract-
01","status":"Success","message":"","events":[{"type":"IED","time":"2015-
01-02T00:00","payoff":-
1000.0,"currency":"USD","nominalValue":1000.0,"nominalRate":0.02,"nominalA
ccrued":0.0},{"type":"IP","time":"2016-01-
02T00:00","payoff":20.0,"currency":"USD","nominalValue":1000.0,"nominalRat
e":0.02,"nominalAccrued":0.0},{"type":"RR","time":"2016-01-
02T00:00","payoff":0.0,"currency":"USD","nominalValue":1000.0,"nominalRate
":0.05,"nominalAccrued":0.0},{"type":"IP","time":"2017-01-
02T00:00","payoff":50.0,"currency":"USD","nominalValue":1000.0,"nominalRat
e":0.05,"nominalAccrued":0.0},{"type":"IP","time":"2018-01-
02T00:00","payoff":50.0,"currency":"USD","nominalValue":1000.0,"nominalRat
e":0.05,"nominalAccrued":0.0},{"type":"RR","time":"2018-01-
02T00:00","payoff":0.0,"currency":"USD","nominalValue":1000.0,"nominalRate
":0.05,"nominalAccrued":0.0},{"type":"IP","time":"2019-01-
02T00:00","payoff":50.0,"currency":"USD","nominalValue":1000.0,"nominalRat
e":0.05,"nominalAccrued":0.0},{"type":"IP","time”:"2020-01-
02T00:00","payoff":50.0,"currency":"USD","nominalValue":1000.0,"nominalRat
e":0.05,"nominalAccrued":0.0},{"type":"MD","time":"2020-01-
02T00:00","payoff":1000.0,"currency":"USD","nominalValue":0.0,"nominalRate
":0.05,"nominalAccrued":0.0}]}][francis@209-133-222-134 actus-curl]
```


Figure 2. Raw JSON stream with returned results of the example /eventsBatch simulation

- The interface(API) to V1.0 actus-webapp/eventsBatch is not publicized formally on the ACTUSfrf web site but, potential ACTUS users can find examples of /eventsBatch

<request> JSON for each ACTUS contract type in the publicly accessible git repository at https://github.com/actusfrf/actus-tests.

4 The need for more general risk factors and a risk factor API

4.1 Risk observation points modelling "risk" events – default, prepayment etc

- Market risk is particularly simple to interface to actus-core cash flow generation
    - Because terms in the ACTUS contract definition explicitly name the Market Object Codes which affect contract behavior.
    - Market risk factor identified with Market Object codes are time series. Their historical values are known; future values can be projected using simple assumptions about market behavior, the business cycle etc.
- Other risk factors, Prepayment, DefaultLoss, Option Exercise, Deposit Account behavior etc are best modelled with more complex risk models parameterized by
    - characteristics of the contract such as industry, loan type, borrower credit rating which do not directly affect cash flow and are therefore not captured in ACTUS contract terms
    - dynamic properties of the economy, markets, etc reflected by specific Marked Object Code values at that time in the simulation
    - Dynamic properties of the contract state – such as its age, the principal outstanding, the current interest rate
    - Counterparty transactional behavior relating to a specific contract
- Different analytic contexts will require different types of risk factor models and different levels of detail within each risk factor. Thee need to be some method for controlling which collection of risk factor models is to be used with each contract while generating its cash flows
- As we noted in the introduction (section 2.1) an important design principle for ACTUS is to maintain a clear separation between the actus-core logic which handles contractual events determined and scheduled by the contract's ACTUS terms and stochastic events such as prepayment or default of a loan.
- The probabilities of these risk factor events can be modelled; the occurrence of the events can be simulated during cash flow even generation BUT the models driving this simulation are "arbitrary" and the choice of the analyst user in contrast to the algorithmic effects on cashflows of the ACTUS contract terms.
- ACTUS deals with this by introducing the concept of *external risk factor observers* – logic which is not part of the actus-core library but is invoked by it at each scheduled *risk observation point* during contract simulation.
- The concept of risk factor observers is outlined in the ACTUS technical specification available at https://www.actusfrf.org/techspecs . Risk observer interfaces ae implemented in the actus-core 1.0 library but there is no published guidance in the technical specification or elsewhere on the specifics of setting up and using risk observer risk factor models.

- Setting variable interest rates in a contract using a market risk factor (we illustrated this with an example in section 3 above), do NOT need to use the risk observer interface. Rate reset events are scheduled by ACTUS terms in the contract specification, so no externally scheduled risk observer events are needed for this. It is the more complex Prepayment, Default, Option exercise, Counterparty behavior risk which need to introduce their schedule of risk Observation points.

4.2 Requirement for a risk factor model store – accessible to ACTUS processing

- A practical scalability and management requirement for modelling risk factors and interfacing them to ACTUS contract simulation is to provide a persistent Risk Factor model store.
- Even the market risk factor indices which can be supplied in line in the webapp/eventsBatch request will benefit from these. If daily market riskfactor data is supplied for long running contracts this becomes quite bulky. Passing this data as a JSON character stream on every contract simulation request incurs serialization, parsing and deserialization costs at the server
- Risk Factor models
- Beyond this there is an organizational scalability issue. In a financial enterprise it it desirable for there to be some centralized repository of risk factor data and risk factor models. This encourages use of consistent updated reference data and basic modes across departments – even when different departments are choosing to use different risk factor modelling for their analytic context.
- Risk models need to be stored as “lookup” data tables rather than risk processing logic. The definition of external risk observer in the ACTUS technical specification leaves open the possibility that the risk observer is provided in the form of some external processing logic. It is however much simpler to store and interface a risk observer provided as pure data – a set of tables, specifying the schedule of risk observation times and some form of multi-dimensional (multi parameter) lookup table or surface for evaluating the risk factor using current contract simulation and market state.

4.3 Scenarios - the aggregated risk factor context for a contract simulation

- Up to this point, we have used terms like risk context and collection of risk Factors to express the ides that an ACTUS contract simulation is run with some specific risk fact environment. We now introduce the term scenario to be the name for a collection of risk factor models which define the complete risk environment for a contract simulation.
- A scenario will include
    - A list of defined market risk factor definitions: each with its Market Object Code and time series of values
    - A list of external risk models each having

        - A risk Factor ID
        - A scheduling rule specifying for each contract where it is applied how to set the risk observation points for that contract
        - A lookup surface specification with the lookup logic to determine the effect at each risk observation point
- Consistency of market risk factors in a scenario
    - The scenario definition includes a list of market risk factor reference index values. Each is a Market Object Code and a time series of values. We would not want one contract in a portfolio to be simulated assuming interest rates are about rise and a different contract in the same portfolio to be simulated with an expectation of falling interest rates. If this were done, it would be invalid to aggregate the portfolio cashflows. So, within the market risk factors of a scenario there is a consistency requirement that there is at most one market risk factor in the scenario list for any supported Market Object Code
- External Risk Model selection for a contract using ACTUS and non ACTUS contract attributes
    - Consistency works differently for external risk models than for market risk factors. The scenario defines a collection of available external risk model – presumed in some sense to be consistent. But a prepayment model which applied to mortgage and loan contracts has no relevance to a SWAP or OPTION Contract. The ACTUS user may want to model default/credit risk for loans in the restaurant business differently from loans in a manufacturing business.
    - Some selection logic has to be available for specifying which set of the available risk models in a scenario are to be used with each contract in simulation using that scenario
    - The most convenient and practical way to achieve this is for contracts to have “user” attributes in addition to their cash flow defining ACTUS standard terms. These contract “user” attributes are then made available to risk factor model selection at the start of contract simulation to set up risk observation schedules and build the risk observer module to be used with that contract.

5 A Risk Factor “Enabled” API for ACTUS cash flow generation

- We propose introduction of an ACTUS Risk Factor Enabled API which would support user commands to:
    - Specify and save in a risk factor store (1) market risk data , (2) multidimensional risk lookup models and (3) scenarios grouping a collection of market risk and risk lookup models
    - Retrieve, edit and delete scenarios, market risk data and risk lookup models from the risk store

    - Request ACTUS cashflow generation for a contract or batch/portfolio of contracts using a named scenario available in the risk factor store as the risk context for the simulation.
- To create and save market risk data, the user provides
    - Its market object code - a unique identifier (riskFactorID)
    - A time series of future and possibly historical values for that risk factor"index"
- To create and save a risk factor lookup model, the user provides:
    - Its Risk Factor ID (riskModelObject code) a unique identifier
    - Specifics on how to generate a schedule of risk observer events for each contract using this model
    - Specification of an n-dimensional lookup surface which will be used in the risk observer for that model to determine the payoff and state transition behavior at each risk event. Lookup parameters can include, time, contract state and available market risk values for that point in simulated time.
- To create and save a scenario, the user provides:
    - A scenarioID uniquely identifying that scenario
    - A list of market risk object codes (Risk Factor IDs) to be used in the scenario
    - A list of Risk Factor lookup Model IDs to be available for contract risk modelling in simulations using the scenario
- When a scenario is defined, all risk factor ID it refers to must already be defined and available in the Risk Factor Store
- To request ACTUS cash flow generation for a portfolio of contracts the user provides:
    - A specification of each contract using ACTUS contract terms
    - The name of the scenario to be used as the risk context for the simulation
    - Additional "user attributes" with each contract definition, indicating which of the risk factor models available in the scenario is to be used with that contract
- The scenario named in a cashflow generation request must be available in the risk Factor store when the request is made.
- Processing of a portfolio simulation request naming a scenario to be used as the risk context, includes the following steps:
    - Scenario data is retrieved from the risk factor store
    - Market risk factor data from the scenario is set up to be available to actus-core library processing
    - For each contract in the portfolio to be simulated
        - the contract events are inserted into the simulation schedule
        - contract user attributes are used to select risk factor models to be used in simulating this contract
        - a risk observer module is constructed for each selected risk factor
        - risk observer events are added to the simulation schedule for that risk factor

- Actus-core is invoked to do cash flow processing for the contract and risk observer event in the contract's event schedule
- At each risk observation event for this contract
  - => call out to the risk observer module
  - => Payoff and State Transition amounts determined by the risk factor lookup

Risk Factor Enabled ACTUS API – Architecture

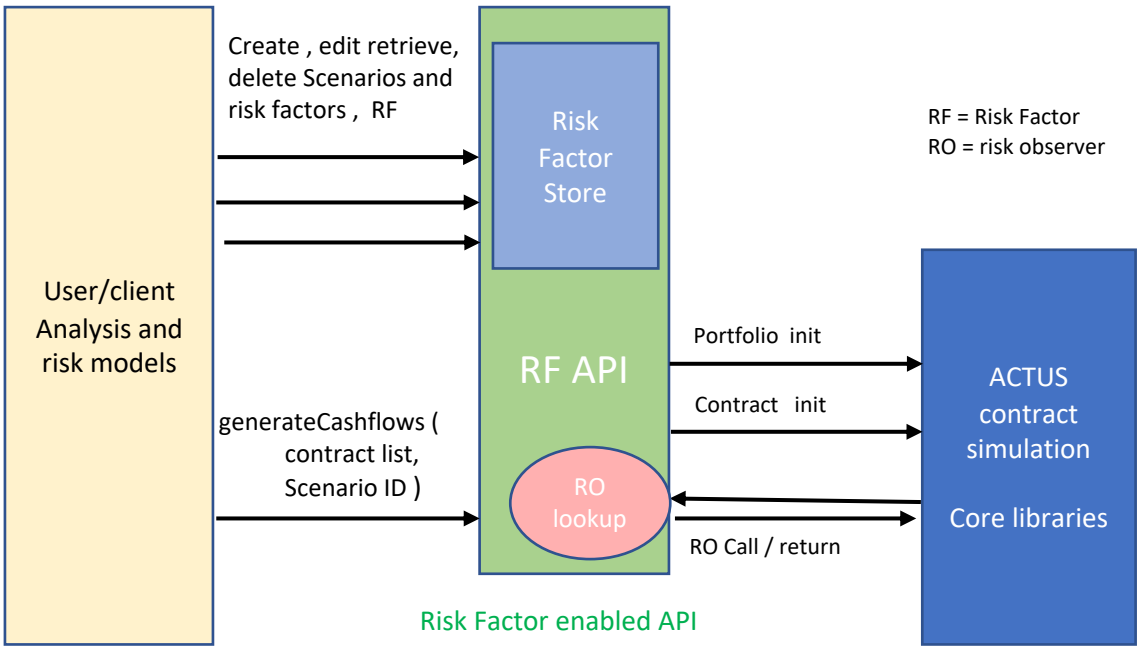

Figure 3 Architecture for a risk factor enabled ACTUS API

- The result returned from a request to simulate a portfolio of contracts using a named scenario as the risk context is a list with an entry for each contract in the portfolio
  - Each contract item includes the contractID and an indication whether cash flow generation for that contract succeeded
  - For each successful contract cash flow generation, the sequence of cash flow events for the contract follows
- The architecture we are proposing for a Risk Factor Enabled API to ACTUS contract simulation and cash flow generation is illustrated in Figure 3 above.

6 Actus-Webapp V1.1 – an initial RF enabled ACTUS API

6.1 Objectives and development plan for the V1.1 webapp implementation

Primary objectives for an initial actus-webapp 1.1 implementation are to provide:

- a first proof-of-concept implementation for a Risk Factor Enabled ACTUS API
- implementation of demand deposit accounts in ACTUS using an UMP (Undefined Maturity Contract) to represent each account with patterns of use deposits and withdrawal amounts to/from their account being handled as transactional data introduced through the risk factor API
    - research colleagues on the DaDFiR3 project required these capability in order to be able to use ACTUS to model stable coin blockchain behavior.
- A start point for documenting and offering a user guide to the SPI (Systems Programmer Interface) for interactions between the Risk Factor Enabled API module and the actus-core library.
    - ACTUS users with Java development skills who need to construct a custom variant of actus-webapp incorporating their own specialized risk factor models, can use the open source Risk factor enabled actus-webapp source code and possibly a user guide commenting on it as documentation for the SPI.

6.2 Starting code (FDIC demo webapp) and V1.1 development Steps

Helpful starting code for implementation of a risk factor enabled ACTUS API was available in the version of actus-webapp developed by Nils Bundi and used in prototype demonstrations of ACTUS capabilities to the FDIC in 2H 2020 and 1H 2021. This version of actus- webapp was never made public and accessible to the ACTUS community but did include partially tested capabilities for:

- Defining and storing scenario data in a persistent mongoDb data store
- Retrieving, editing and deleting this risk factor data
- Requesting simulation for a list of contracts using an available named risk scenario

Development tasks in evolving this code into actus-webapp V1.1 included :

- Storing a version of the FDIC-webapp code in the https://github.com/actusfrf/actus-webapp where it is visible to the ACTUS community and available for review pull requests, etc
    - A version of the code supporting a Prepayment model with some updates to allow deposit transactions and compatible with the original FDIC format scenario data store is saved in branch fdicV101PP in the repository
    - The most recent version of actus-webapp V1.1 which includes a Deposit Transaction risk model class and supports deposit and withdrawal transactions specified as a risk factor lookup surface. The scenario data store

format is changed from the FDIC version to allow DepositTrx risk models to be included in a scenario

- Rebuilding the FDICwebapp using a version of the actus-core jar which is updated to allow variable interest rate UMP contracts
    - The version of actus-core which supports variable rate UMP contracts is saved in https://github.com/actusfrf/actus-core in the branch UMPwithVR
    - Important to use this since we want demand deposit UMP accounts which allow variable interest rates.
- Recovering the FDIC scenarios and validating/fixing the prepayment capabilities of this code.
    - The mongodb database of scenarios used in the FDIC demo was still available with scenarios
    - A curl request for simulation of an ANN contract was use to test prepayment
- Classes TwoDimensionalDepositTrxModel and TwoDimensionalDepositTrxModel Data were developed to provide the logic for processing deposit/withdrawal transactions into an UMP contract, and to allow scenarios to include TwoDimensionalDepositTrxModels
- Classes for LabelMargins and LabelSurface were developed to allow risk observer lookup where the input parameters are text strings (in the case of DepostTrx models these are contractIDs and date strings) rather than the numeric parameters supported previously with Margin and Surface classes.
- Test curl command samples were developed:
    - To create a scenario including a sample DepositTrx model
    - To request cash flow generation for an UMP contract with deposits and withdrawals

6.3 Webapp V1.1* System Diagram and processing overview

Figure 4 below is a system diagram for the proposed actus-webapp V1.1. It follows the Risk Factor enabled API architecture of Figure 3 but adds some more detail.

- This version of webapp uses a Mongodb database to store scenario and risk factor data persistently
- Webapp entry point /scenario/save takes as input a JSON scenario specification and saves this information in the scenario and risk factor store keyed by the scenarioID
- Additional webapp /scenario entry points provide commands to retrieve, update, and delete scenarios using scenarioID as an input parameter to identify which scenario to operate on
- Webapp entry point /simulation/runScenario takes as inputs:
    - A JSON list of one or more contracts – each specified as a list of ACTUS term values and contract user attributes selecting risk models to be used with this contract
    - A scenario ID – identifying the risk context for this simulation

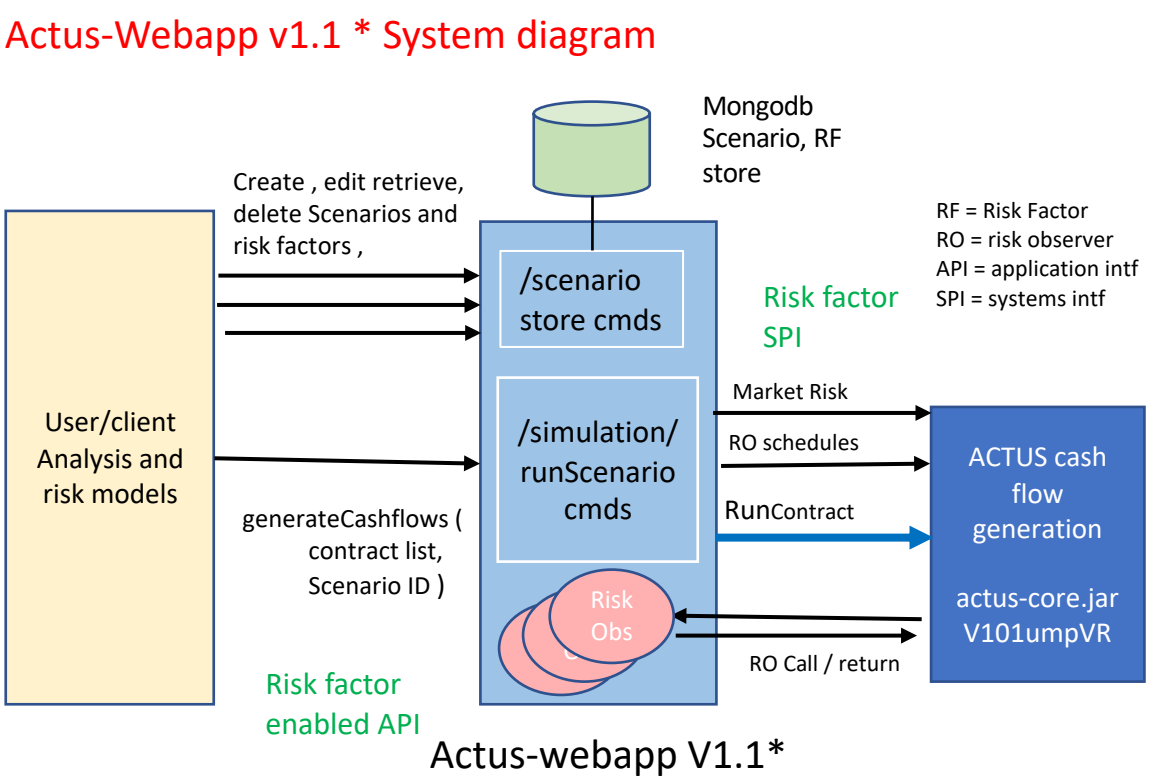

Figure 4 System diagram for actus-webap1.1 (proposed) system showing interactions with other system components

- In webapp processing of a /simulation/runScenario request the following interactions with other system components occurs:
    - Data for the selected scenario is retrieved from the mongodb risk factor store. An error can occur if the named scenario is not found
    - Market Risk data for the simulation is extracted from this scenario and organized as a marketRisk object
    - For each contract in the input contract list
        - Using contract “risk model selector” user attributes, a list of risk models from the scenario to be used with this contract is created. An error can occur if a delected risk model is not found in the scenario
        - For each selected riskModel, a schedule of risk observer events is created and added to the base contract event schedule for the contract.
        - For each selected risk model, a risk observer created is created using scenario specification of the risk model to construct the risk lookup surface

- Webapp then calls the runContract entry point in the actus-core. jar passing the market risk and updated RO event schedule.
- During the resulting cashflow event generation for the contract there can be multiple risk observer call outs from actus-core to a specific risk observer object, to obtain the risk outcome for the contract at that point in simulated time.
- When cash flow generation for this contract is complete, actus-core returns the generated list of cash flow events
- Webapp appends this into its output list of cash flows for each contract.

6.4 Webapp V1.1 Scenario Definition , JSON Data structure and descriptive notes

In Webapp V1.1 the JSON data which defines a scenario and is the expected input parameter format for /scenario/save is :

<scenario-def> ::= { *<scenarioID-def>* , *<market-risk-data>* , *<term-structure-data>*,

<2Dprepayment-models> , *<2Dcredit-loss-models>*,

<2DDepositTransactionModels>

}

A scenario definition specifies: its scenario ID, optionally some market risk factors, optionally some 2 parameter prepayment models, optionally some 2 parameter credit loss models and optionally some 2parameter Deposit/Withdrawal transaction models. The definition is organized as a standards JSON comma separated list of name-value pairs enclosed in curly braces { } .

<scenarioID-def> ::= "scenarioId" : ID-STRING

The scenario is identified by an ID-STRING. This must be alphanumeric (case sensitive) string starting with an alphabetic character. Scenario IDs need to be unique in the working namespace of scenarioIDs, riskFactorIDs, and marketObjectCodes.

<*market-risk-data*> ::= “timeSeriesData” : [<*market-risk-factor-def*>. . .]

<*market-risk-factor-def*> ::= { “marketObjectCode” : ID-STRING , “base” : NUMBER,

“data” : [< *time-series-data-item*> . . .]

}

<*time-series-data-item*> ::= { “time” : DATETIME-STRING , “value” : NUMBER }

The scenario definition optionally includes definitions for some number of market risk factors. Each mark risk fact is defined by specifying its ID-STRING marketObjectCode. (marketObjectCodes are a subset of riskFactorIDs). The base of the market risk factor is a number – typically 1.0 or 100.0 . A base of 100.0 indicates that that values for the market risk factor are %ge (interest) rates. With this base, a market risk factor value or 4.0 is interpreted as a 4% pa Rate. The values of the market risk factor are specified as a time series – a list of one or more data items each specifiying the actual or projected value of the market object code at a designated date-time. The format for date-times is “yyyy-mm-ddT00:00:00”; the value of a time series data item is set as a number i.e. 3.6 , 100.0 , -4.2.

We see that format for defining marketObjectCodes and marker risk factor data, exacly matches the *inline* definition of market risk data in a version 1.0 webapp/eventsBatch request. In a scenario definition this information is stored in the risk factor store. This allows it to be referenced in a version 1.1 webapp/simulation/runScenario request and avoids having to include possibly bulky reference index projects on every simulation request.

<*term-structure-data*> ::= “termStructureData” : [ ]

Term structure data is not currently used in actus-webapp v1.1 but must be present in scenario definitions as an empty list as above

<*2Dprepayment-models*> ::=

“twoDimensionalPrepaymentModelData” : [<*2Dprepayment- model-def*> …]

<*2Dprepayment-model-def*> ::= { “riskFactorID” : ID-STRING,

“referenceRateId” : ID-STRING,

“prepaymentEventTimes “: [TIMEDATE-STRING . . .] ,

“surface” : { *<2D surface-def >* }

}

< 2D surface-def> ::= { “interpolationMethod” : “NA” , “extrapolationMethod” : “ NA”,

“margins” : [{ “dimension” : 1 , “values” : [NUMBER . . .] },

{ “dimension” : 2, “values” : [NUMBER . . .] }

] ,

“ data” : [[NUMBER . . .] . . .]

}

In addition to defining market risk factor data, a scenario can introduce a number of “lookup surface“ risk factor models which will be realized as ACTUS external risk observer processing. Categories of external look up risk model supported in ACTUS webapp 1.1 are:

- Prepayment Models – using: (1) loan age and (2) spread between loan rate and current market interest rates as input parameters and returning the “fraction of outstanding principal prepaid” at this risk event
- Credit Default Loss models – using: (1) loan product category (2) loan credit rating as input parameters and returning the fraction of outstanding principal lost due to default at this risk event.
- Deposit and Withdrawal transaction amounts on a demand deposit account (modelled as an UMP contract) – using: (1) account id, and (2) transaction date as input parameters and returning the actual amount deposited or withdrawn on that date.

We will describe the syntax of a Two Dimensional prepayment Model definition here and comment later on the differences seen in definitions for the other two model categories.

The definition of a 2D prepayment model includes a declaration of its riskFactorId – an ID-STRING uniquely identifying this risk factor. It also defines a referenceRateID. The risk observer processing for this model will include a step to get the current-simulation -time value for this reference rate and use that is calculating the spread between the current loan rate and the market rate which would be relevant if the loan is refinanced. The referenceRateID in a prepayment model must match a marketObjectCode in the containing scenario’s <*market-risk-data*> a list to make sure that values for the reference Rate will be available.

The definition of a Two Dimensional Prepayment Model also includes a list of prepaymentEventTimes. These are date-time strings in “yyyy-mm-ddT00:00:00” format setting the schedule for risk observer events when a contract is simulated. They are specified explicitly.

Amore general model definition might allow “rules” such as – quarterly for the life of the contract.

The final item in a 2D prepayment model definition is the specification of the 2D lookup surface for the model.

A 2D Surface Definition includes terms to set interPolationMethod and extrapolatioNMethod. Webapp-Version1.1. does not support this capability so both terms need to be set to the value “NA”. Next there is a specification of the margin values for each of the dimensions of the two-dimensional surface. In our 2D Prepayment models, where the input parameters are loan age and loan-to-market rate spread, these are lists of numeric values. The margin values define the grid of lookup input parameters. The remaining term in a 2D surface definition is the data set of lookup surface values. These are specified as a list of lists of numeric output values. For a 2D surface the data is provided as a list of row data value lists. The row data value lists have the same number of values as dimension2 label values; the number of row data value lists matches the number of dimension1 label values.

The format of a 2D creditDefaultLoss Model is essentially identical to that for a 2Dprepayment Model.

The format for a 2D depositTransaction model is different because the input parameters for lookup using this model are contractIDs (accountIDs) and transactionDates which will be DATETIME-STRINGS. There is some redundancy in having to specify a list of depositTrxEventDates for the model and then actual transaction dates for each account transaction. TO keep things simple the depositTrxEventDates should be the union of different Transaction dates over all the accounts in the model.

In DepositTrx Models , the output from a risk event lookup will be the amount of the deposit. A negative amount returned will be a withdrawal. Thus positive deposits increase the principal of the contract – increasing liability of the liability counter party and assets of the asset counterparty. More details on that and logic to avoid “overdrawing” a demand deposit account will discussed in the following section – design overview of the webapp1.1 implementation

6.5 Webapp V 1.1 design overview

Outline

Entry points / requests

- In addition to the demo, demo-meta, events and eventsBatch request supported in V1.0 webapp . webapp V1.1 add the following entrypoint/requests types with new controllers
- Class ScenariosController adds request types
    - /scenario/save

    - /scenario/delete
    -
- Class SimulationController adds the new request /simulation/runScenario which takes as input a list of Contracts and a scenario id and generates the list of output cashflow sequences

Processing logic outline for /simulation/runscenario

- Method xxx gets attributes of request
- Retrieves scenario data for selected scenario
- Creates market data object
- Creates empty output list
- Iterates through list of ATUS contracts in input: for each contract
    - Create risk observers for this contract
    - For each model type: (prepayment, CreditDefault, DepositTrx)
    - Looks in contract for user attributes selecting a specific riskModel
    - Checks in scenario for risk model with that riskFactorID,
    - Creates a risk observer for that riskactus-core function
- Create the fully populated event list
- Start with an empty event scehdule
- Call actus-core (passing contract object to add ACTUS basic contract events

7 Running a sample UMP Contract with Deposit Transactions

7.1 Outline

We describe the action of a sample Deposit transaction risk factor on a sample UMP "demand deposit account" contract as an illustration.

Part of the motivation for this document is to provide a "User guide" for a potential user interested in the interaction of ACTUS contracts with risk models. Describing the behavior of some simple curl example requests May be a useful primer.

A series of steps is described:

- Building the actus-core library which does the actusl cash flow computation
- Building the code for the actus-webapp V1.1 web service using actus-core.jar
- Configuring and starting an actus-webapp V1.1 service
- Creating an example scenario with a DespositTrx risk factor model
- Deleting the example scenario from the risk factor store

- Running an example UMP contract simulation using the DepositTrx risk factor model in the example scenario

7.2 Building the actus-core library (jar) file

- Establish a home working directory HOME
- Download or clone from https://github.com/actusfrf/actus-core branch = V101umpVR into directory HOME/actus-core
    - this version supports variable rate ump contracts
- cd to the directory HOME/actus-core
- use maven to build and make generated jar available with command:
    - > mvn install clean
        - Prereqs for building actus-core are described in the https://github.com/actusfrf/actus-core/README
- This mvn command should process and report success
    - The above step actually backtests the new jar against a list of example tests for which correct output is known

7.3 Configure, build and launch the actus-webapp Version1.0 application

- Assuming that the actus-core.jar is already build following the steps in the preceding subsection ….
- .. clone or down load from https://github.com/actusfrf/actus-webapp branch = fdicV101DW , into HOME/actus-webapp
- It is possible that you may want to change the configuration for your server
    - Modify the port on which the actus-webapp server runs
    - Change the mongodb which it will use to find demo examples and use as its fiskfactor store
- Inspect HOME/actus-webapp/scr/main/resouces/application.setting
- Text edits to the values assigned for port and mongodb database name will configure these settings
- Prereqs for for building and running actus-webapp are described in https://github.com/actusfrf/actus-webapp/README
- To build the actus-webapp code :
    - > cd HOME/actus-weball
    - > ./gradlew build
- The above gradle request, will process for a while build the application, run some regression tests and report any errors.
- If the actus-webapp build was successful, the actus-webapp service can now be started with the command

    - > ./gradlew bootRun
- If issued as a synchronous command, this will show a tomcat server being started on the configured local port waiting for request.
- Benefits of starting the server synchronously while in experimentation mode are:
    - Trace for any errors in process curl requests will show in the console
    - Diagnostic prints from this (alpha) version actus-webapp will show on the console
    - The server can be stopped at any time by issuing CntrlC on the command line – allowing easy, edit rebuild, relaunch

7.4 Creating a Scenario with a Deposit/withholding risk model

- There is a directory of sample curl request files in HOME/actus-webapp/sampleCurls
- The examples in this directory are configured to work with a webapp server running locally on port 8083; if your web server is running on some other port, edit the target url for any of these examples to your port#
- The command:
    - > source HOME/actus-webapp/sampleCurls/f3addScenDeposits.txt
    - … will execute a request to create scenario with scenarioID = “Deposits”
    - => an actus-webapp V1.1 server needs to be running on localhost:8083 for this to work; follow guidance of preceding subsections to start this
- This scenario includes definitions for:
    - A single interest market risk factor riskFactorID/marketObjectCode = PRIME with the following table of time series data

Date	*Rate % pa*
2022-12-15	3%
2023-12-15	4%
2024-12-15	5%
2025-12-15	4.5%

    - No prepayment models
    - No Credit Default loss models
    - A single TwoDimensional DepositTrx Risk model with riskFactorID=”DEPOSIT_TRXS” and the following table of deposit and withdrawal amounts on different dates to two UMP demand deposit accounts with contractIDs ump001 and ump002:

Date	*"ump001" amount*	*"ump002" amount*
2023-06-01	+ 1.5	+ 1.5
2024-06-01	+ 20	- 0.8
2025-06-01	+ 100	- 150

- The deposit and withdrawal amounts are assumed to be in the currency of the contract
    - Probably it would be better to have a default currency for the scenario and require all contracts using the scenario to be defined in scenario default currency terms. That requirement would ensure that aggregated portfolio cashflow results are meaningful
    - This was not an issue for the prepayment and credit default loss where the risk model output is a dimensionless ratio; the notional capital of the contract is reduced by this ratio.
    - For transactional risk model behaviors being applied to individual contracts, the risk model output does need to be a specific +ve or -ve amount in a defined currency.
    - For analytic modelling of deposit and withdrawal behavior on blocks of (similar) accounts, it may be more convenient to work with a dimensionless ratio change as output from the model

7.5 Deleting or Editing a scenario from the Risk factor store

- An existing scenario with scenarioID = "Deposits" can be deleted using the sample curl request :
    - > source HOME/actus-webapp/sampleCurls/f3deleteScenDeposit.txt
- The text of this request file is:
- ```
  curl –L ––request POST
  http://localhost:8083/scenarios/deleteScenario –H content–
  Type:application/json ––data–raw '{"scenarioId" : "Deposits"}'
  ```

- A sample file HOME/actus-webapp/sampleCurls/f3deleteScenAll.txt is available to clear the risk factor store of all currently defined scenarios.
- This command should be used with caution is there are multiple users of the risk factor store

7.6 Simulation of an UMP contract with "risk factor" Deposit/Withdrawal transactions

- Sample file HOME/actus-webapp/f3UMPnRFyDX.txt is a curl request for ACTUS cash simulation of an UMP contract with contractID "ump001" using scenario "DEPOSITS" for the risk factors.
  ```

- Previous subsection described how to create the Scenario with scenarioID = “DEPOSITS” and have this saved in the Risk Factor store
- This scenario includes a DepositsTrx model with riskFactorID = “DEPOSIT_TRXS” which defines deposit and withdrawal amounts and dates for contract with contractID = “ump001”
- This sample curl request file
    - Is a request to the web service at: localhost:8083/simulation/runScenario
    - To be run with scenario “Deposits”
    - To simulate until “2027-01-01”
        - this is neede because UMP contract have no specific maturity date
        - The simulation needs to know when to stop
    - The contract list consists of a single contract with contractID = “ump001”
- Pretty printed version of the JSON defining contract “ump001” - from the file f3UMPnRFyDX.txt is:


```
 "calendar":"WEEKDAY",
"businessDayConvention":"SCF",
"contractType":"UMP",
"statusDate":"2022-02-28T00:00:00",
"contractRole":"RPL",
"contractID":"ump001",
"cycleAnchorDateOfInterestPayment":"2022-03-01T00:00:00",
"cycleOfInterestPayment":"P6ML0",
"nominalInterestRate":0.02,
"dayCountConvention":"30E360",
"currency":"USD",
"contractDealDate":"2022-02-28 T00:00:00",
"initialExchangeDate":"2022-03-02T00:00:00",
"maturityDate":"2026-12-01T00:00:00",
"notionalPrincipal":1000,
"premiumDiscountAtIED":0,
"objectCodeOfCashBalanceModel": "DEPOSIT_TRXS"
```


- This is an UMP contract with contractID = “ump001”
- The account was opened (InitialExchangeDate) 2022-03-02
- With an opening balance (notional principal) of $1000 USD
- Accrued interest at the status date defaults to $0
- Interest is paid at a fixed rate at 6 monthly intervals
- The role is liability i.e. this is the bank’s view not the account holder’s ;
    - The account holder sees it as an asset and may receive interest
- A user defined (risk) contract attribute objectCodeOfCashBalanceModel = “DEPOST_TRXS”
    - This identifies the riskFactor model to be used when doing cash flow generation for this contract
    - => a better attribute name might be “riskFactorIdOfDepositTrxModel”
    - => see list of candidate code improvements below
- Issuing the command
    - > source HOME/actus-webapp/sampleCurls/f3UMPnRF.txt

    - with an actus-webapp V1.1 service running at localhost:8083
- .. will run a cash flow generation for this contract

The header returned from this request is:

```
[{ "scenarioId":      "Deposits",
   "contractID":     "ump001",
   "status":          "Success",
   "message":         "" ,
   "events":          [{" … JSON list of ump001 events …
```

7.6.1 Table of cash flow events returned from the simulation

Below is a table showing the events returned from ump001 simulation is:

Tx#	type	Time	payoff	Curr-ency	Nominal value	Nominal rate	Nominal accrued
1	IED	2022-03-01	1000	USD	-1000.0	0.02	0.0
2	IPCI	2022-03-01	0.0	USD	-1000.0	0.02	0.0
3	IPCI	2022-09-01	0.0	USD	-1010.0	0.02	0.0
4	IPCI	2023-03-01	0.0	USD	-1020.10	0.02	0.0
5	PR	2023-06-01	1.50	USD	-1021.60	0.02	-5.1005
6	IPCI	2023-09-01	0.00	USD	-1031.81	0.02	0.0
7	IPCI	2024-03-01	0.0	USD	-1042.13	0.02	0.0
8	PR	2024-06-01	20.00	USD	-1062.13	0.02	-5.21
9	IPCI	2024-09-01	0.0	USD	-1072.65	0.02	0.0
10	IPCI	2025-03-01	0.0	USD	-1083.37	0.02	0.0
11	PR	2025-06-01	100.00	USD	-1183.37	0.02	-5.42
12	IPCI	2025-09-01	0.0	USD	-1194.71	0.02	0.0
13	IPCI	2026-03-01	0.0	USD	-1206.65	0.02	0.0

7.6.2 Commentary on these cash flow events

- Event 1 is the InitialExchangeDate – the account is opened with an initial deposit of $1000 on 2022-03-01
    - This amount shows as a +ve deposit payoff amount increasing the conceptual notional principal of the account
    - Since we are showing the liability view of the contract – the view from the Bank offering the demand Deposit account to a user – the nominal value is -$1000 after the initial deposit is made
    - The nominal interest rate is 2% pa fixed
    - There is no accrued interest
- Event 2 is a IPCI – Capitalized Interest Payment occurring on the same day
    - There is no accrued interest because the account has just been created

    - This null IPCI could have been avoided by setting the cycleAnchorDateOfInterestPayment in the contract = “2022-09-01” – ie.to the first date on which interest will actually be paid
    - This is probably a better ACTUS encoding for this contract – and usual practice
- Event 3 is the first effective capitalized interest IPCI event on 2022-09-01
    - The payable interest 6 months on $1000 at 2% pa is $10
    - Since the interest is capitalized, the payoff is $0
    - But the notionalValue (liability) of the bank issuing the account “increases” from $ -1000.00 to $ -1010.00
    - Accrued interest on completion of this event is still $0.00 because the interest has been capitalized.
- Event 4 is another IPCI event occurring on 2023-03-01
    - Note that interest is being calculated correctly; the notional value has gone form $- 1000 to $ -1020.10 reflecting 2% pa interest rate with 6 month compounding
- Event 5 is the first Deposit Transaction made by the account holder occurring on 2023-06-01
    - The payoff is $ 1.50. – a small positive deposit – this amount is cash inflow to the bank
    - The Liability of the bank increases by this amount; the notional value changes s from $ -1020.1 to $ -1021.6
    - Since we in between interest capitalization events there is accrued interest liability of $ -5.01 at this point. The increased nominal value in the account will affect accrued interest for the period 2023-06-01 until the next IPCI event at 2023-09-01
- Event 6 is the next IPCI event

The remaining events in the generated cash flow follow the same pattern as above – a mix of IPCI and PR events. We continue to call deposits PR (Principal Reduction) events even though the principal in the account is *increased* by each deposit transaction. Withdrawals will be demonstrated in a following section.

Principal Change might be a better name for the event type – but use of Principal Reduction is established by historical usage.

7.7 Simulating the Asset View of the UMP transaction in f3UMPnRFyDX.txt

In HOME/actus-webapp/sampleCurls , the file f3UMPnRFyDXa.txt is a request to simulate what is effective the account holders view of the same ump001 contract. The changes made to the ACTUS terms for the contract (view) are:

- contractRole = RPA -- the asset (account holder’view)
- contractID = ump001 -- no change in the contractID
- cycleAnchorDateOfInterestPayment = 2023-03-01

    - setting this to occur a full period after IED avoid the null IPCI event of the previous Liability side example
- cycleOfInterestPayment = P1YL0
    - interest payments once a year will mean less compounding for the account holder but fewer events in total – so easier to review the effects of Deposit transactions
- All other terms are the same as in the previous (liability side) example

Running the curl request of f3UMPnRFyDXa, is a /simulation/runScenario request and returns the following table of cashflow events.

Tx#	type	Time	payoff	Curr-ency	Nominal value	Nominal rate	Nominal accrued
1	IED	2022-03-01	-1000.00	USD	1000.0	0.02	0.0
2	IPCI	2023-03-01	0.0	USD	1020.00	0.02	0.0
3	PR	2023-06-01	-1.50	USD	1021.50	0.02	5.1
4	IPCI	2024-03-01	0.0	USD	1041.92	0.02	0.0
5	PR	2024-06-01	-20.00	USD	1062.92	0.02	5.21
6	IPCI	2025-03-01	0.0	USD	1083.06	0.02	0.0
7	PR	2025-06-01	-100.00	USD	1183.06	0.02	5.41

Commentary on these cash flow events

- Event 1 is the IED as before with the differences that:
    - The payoff is negative $ -1000; account holder deposits $1000 when the account is opened making these funds available to the bank “offering” the demand deposit account.
    - The Nominal value is $ +1000.00 – an asset for the account holder
- Event 2 is the first IPCI event occurring a full year after the account is opened
    - The Capitalized interest for this event is exactly $20.00 because there was no compounding during the year resulting from the change setting cycleOfInterestPayment to 1 year
    - The nominal value in this account is increased by exactly this capitalized interest amount to $ 1020.00
- Event 3 is an account holder deposit transaction of $1.50, occurring on 2023-06-01 pa
    - This shows as a negative payoff - the account holder is depositing additional funds
    - But the notional value of the account increases to $1021.50
    - The accrued interest is positive. We are between IPCI event and this is an asset which will eventually be capitalized into the account
- Event 4 is another IPCI and subsequent events 5, 6 and 7 follow the same pattern.

7.8 A demand Deposit account UMP with withdrawals - test insufficient funds logic

File HOME/actus-webapp/sampleCurls/f3UMPnRFyDXW.txt has a sample request to simulate an UMP demand deposit account contract with the following properties:

- ContractID = “ump002”
  - In the DEPOSIT_TRX risk surface there are withdrawals for this contractID
  - The larger withdrawal is for $150
  - By making the notional balance in the account $100 we can test the cash flow events generated for a withdrawal request which exceeds th current balance in the account
- NotionalPrincipal = $100.00
  - This wilo set up the insufficient funds situation described above
- Contract role = RPL
  - this is the liability view i.e the view of the bank offering deposit accounts , not the view of the customer using the account
- All other parameters correspond to the “simplified case – the customer view of the ump001 account captured in f3UMPnRFyDXa.txt

The cash flow event sequence returned for this request is captured in the table below.

Tx#	type	Time	Payoff	Curr-ency	Nominal value	Nominal rate	Nominal accrued
1	IED	2022-03-01	100.00	USD	-100.00	0.02	0.0
2	IPCI	2023-03-01	0.0	USD	-102.00	0.02	0.0
3	PR	2023-06-01	1.50	USD	-103.50	0.02	-0.51
4	IPCI	2024-03-01	0.0	USD	-105.56	0.02	0.0
5	PR	2024-06-01	-0.80	USD	-104.86	0.02	-0.53
6	IPCI	2025-03-01	0.0	USD	-106.86	0.02	0.0
7	PR	2025-06-01	-106.86	USD	0.0	0.02	-0.53

From the perspective of the bank

- the first principal change is event 3 at 2023-06-01 is a deposit with a positive payoff of + $1.50 which increases the banks liability to $-103.5
- The next principal change in event 5 is a withdrawal seen be the bank as a negative payoff of $-0.80 decreasing its liability to $-104.86
- The final principal change in event 7 was actually a withdrawal request for $150 (this can be seen from the DEPOST_TRX lookup surface . Since this is more than than the available funds at that time, the actual amount paid out is just the available balance of $106.86 and the banks liability on the account goes to $0.0
- Interest is paid in IPCI events as previously. This is shown as liabilities of the bank.