13th - 17th May 2019 – Kuala Lumpur

Fraud Model Development and Deployment in SAS FM

Session 3: Signatures

Signatures

Topics

- Introduction
- Signature definition
- Fetching signatures
- Initializing signatures
- Updating signatures
- Applications
- Upgrading and downgrading signatures
- Design considerations

Signatures

Introduction

Signatures

Illustrative Example: Behavioral Profiling

Peter is paying Paul $10 using his mobile app. Is the transaction suspicious?

Mobile Device

Have we seen this device before?

Is this device only associated with Peter?

Is it blacklisted?

Peter's Customer ID

Has Peter transacted with Paul before?

Does he usually send money at this time of the day?

Does he use the same account for payments?

Any recent password changes?

Paul

Has anyone sent money to Paul before?

How long have we known Paul?

What is the typical amount Paul receives?

Signatures

Illustration: Card Level Signature

Date / Time		Amount	MCC	Merchant	Zip Code	POS
20140301	09:27:10	77.49	5542	MARATHON #1567	92137	02
20140302	13:23:01	102.08	5713	BANANA INTERNATIONAL PLA	92122	90
20140303	12:51:35	40.54	5713	OLD NAVY-ON DRCT ONLNE	94105	01
20140304	18:12:55	45.22	5542	CHEVRON UTC	92122	90
20140304	18:46:20	31.19	5651	ANTHROPOLOGIE #569	92122	90
20140304	19:21:43	42.12	5712	THE LAND OF NOD 158	92122	90
20140304	19:38:25	2.75	5814	CHAMPAGNE BAKERY 4207	92122	05
20140306	15:20:07	78.56	5713	AMAZON RETAIL	92122	81
20140307	08:38:45	1.00	5542	MARATHON PETRO041350	92064	02

-- Raw transactional data captured in a card signature --

Signatures

Illustration: Card Level Signature

Date / Time		Amount	MCC	Merchant	Zip Code	POS
20140301	09:27:10	77.49	5542	MARATHON #1567	92137	02
20140302	13:23:01	102.08	5713	BANANA INTERNATIONAL PLA	92122	90
20140303	12:51:35	40.54	5713	OLD NAVY-ON DRCT ONLNE	94105	01
20140304	18:12:55	45.22	5542	CHEVRON UTC	92122	90
20140304	18:46:20	31.19	5651	ANTHROPOLOGIE #569	92122	90
20140304	19:21:43	42.12	5712	THE LAND OF NOD 158	92122	90
20140304	19:38:25	2.75	5814	CHAMPAGNE BAKERY 4207	92122	05
20140306	15:20:07	78.56	5713	AMAZON RETAIL	92122	81
20140307	08:38:45	1.00	5542	MARATHON PETRO041350	92064	02

Average velocity of spend

Used this card for gas before?

Average spend at gas stations

Been to this brand before?

Transaction distance velocity

Propensity for swiped transactions

Signatures

So how do we actually store this information in the signature?

Date / Time	Amount	MCC	Merchant	Zip Code	POS
20140301 09:27:10	77.49	5542	MARATHON #1567	92137	02
20140302 13:23:01	102.08	5713	BANANA INTERNATIONAL PLA	92122	90
20140303 12:51:35	40.54	5713	OLD NAVY-ON DRCT ONLNE	94105	01
20140304 18:12:55	45.22	5542	CHEVRON UTC	92122	90
20140304 18:46:20	31.19	5651	ANTHROPOLOGIE #569	92122	90
20140304 19:21:43	42.12	5712	THE LAND OF NOD 158	92122	90
20140304 19:38:25	2.75	5814	CHAMPAGNE BAKERY 4207	92122	05
20140306 15:20:07	78.56	5713	AMAZON RETAIL	92122	81
20140307 08:38:45	1.00	5542	MARATHON PETRO041350	92064	02

Separate 1-D arrays whose elements are in ‘sync’

Signatures

Illustration: Card Level Signature

Date / time of last address change	20140301 10:50:00
Available credit limit	1500.00
Average amount on ATM transactions	42.55
Number of ATM transactions	25

-- Summary / aggregate / scalar data captured in a card signature --

Signatures

Introduction

- Signatures provides a mechanism to convert entity behavior into inputs for the model
- Can store raw transactional data to minimize information loss
  - Selected elements of new transactions get inserted into the signatures
    - Elements chosen for storage are determined during the analytic design phase
- Can also store summarized / aggregate information in various forms
- Provides the ability to form a holistic view of information originating from different channels including financial and non-monetary activities

Signatures

Introduction

- Information can be maintained at up to 16 different entity levels
  - However having too many signatures can impact throughput performance
- Real-time accessibility; is read and written to in real-time
  - Signatures are fetched from the database and made available to the model along with the incoming transaction
  - Models update the signature and are written back to the database after processing the transaction

Signatures

Signature Definition

Signatures

Defining a Signature

- A signature is defined by a set of `length` statements that specify the type (character or numeric) and length of each individual variable
- Array elements are decomposed into its individual scalar components +

```
length z00_datetimes_1 - z00_datetimes_5          8.;
length z00_amount_1 - z00_amounts_5               8.;

length z00_last_address_change_dt                 8.;
length z00_credit_limit                           8.;
length z00_average_atm_amount                     8.;
length z00_count_atm_txns                         8.;
length z00_arr_curlength                          8.;
length z00_arr_curr_index                         8.;
length z00_arr_prev_index                         8.;

length z00_mccs_1 - z00_mccs_5                    $4.;
length z00_zipcodes_1 - z00_zipcodes_5            $5.;
length z00_merchants_1 - z00_merchants_5          $40.;
length z00_poss_1 - z00_poss_5                    $2.;
```

Optional bookkeeping variables discussed in detail later

+ In SAS, an array is not a data structure but is just a convenient way of temporarily identifying a group of scalars.

Signatures

Signature Storage

- Signatures are stored in the ‘Z table’ within the multi-entity history database (MEH)
- Each signature instance is stored as a single row in the MEH

SEGMENT NO.	LOOKUP KEY	CONTENT_ID_VERSION	CONTENTS
< 00 - 15 >	100 Bytes	<8 BYTE CONTENT ID><4 DIGIT VERSION>	Signature contents parsed into a binary blob
00	4000ABCDEFGHIJKL	CARD_EG_0100	.....

Row structure of a signature record +

+ Only the four pertinent elements are shown for brevity; there are additional elements as part of the signature record

Signatures

Signature Storage

- Segment number indicates the signature segment to which the entry corresponds to.
  - Given SAS FM supports 16 different segments, the values range from 00 to 15
- The lookup key holds the actual entity value for which the record corresponds to.
  - E.g., in a card signature, this would contain the actual card number
- The contents field contain all the actual signature content
  - The various individual component scalars are parsed into a binary blob and stored as one single field
  - There are limits on the size of the contents depending on the platform (Z/OS vs. Linux) and database used for the MEH (Oracle vs. DB2)
- Segment no. concatenated with the lookup key serves as the primary key+

+ There are other components to the primary key; but this definition suffices for the scope of our discussion

Signatures

CONTENT ID VERSION (CID)

- The Z##_CONTENT_ID_VERSION field serves as a versioning mechanism
- It is very common for the signature definition to be changed when models are rebuilt
  - New elements may be added
  - Existing elements may be removed
  - Arrays be expanded or shrunk
  - Occasionally the key definition may also change
  - In very rare occasions a segment may be fully retired and reclaimed back to be used on a different entity
- This leads to the structure of the signature content to change as well
  - The process of converting the individual signature elements to a binary blob relies on the knowledge of which elements
  - Therefore the system relies on the CID properly parse this blob when a signature is read from or written to the MEH

Signatures
CONTENT ID VERSION

- The first 8 bytes is the signature ‘Family Name’. E.g.
  - CARD_EG_
  - BANKCUST
  - BENEACCT
- The next 4 bytes denote the actual version.
  - Typically first 2 bytes denote the major version and last 2 bytes the minor version
  - E.g. 0100, 0101, 0200
- Only a single CID can exist for a given key on a given segment
  - Recall that segment no. concatenated with the lookup key serves as the primary key; CID is not part of that primary key

Signatures

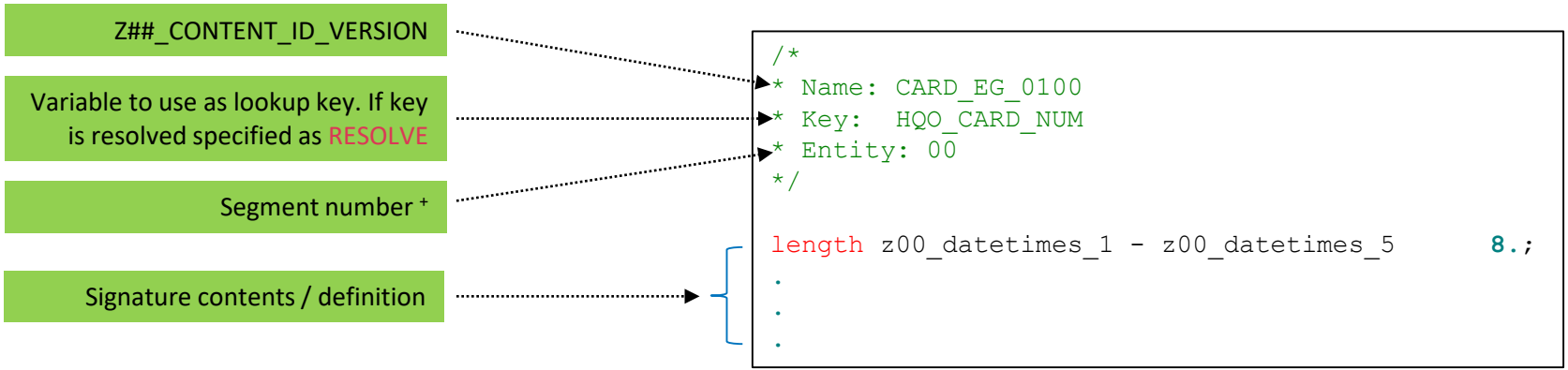
CONTENT ID VERSION

- If a signature lookup does not return a row, CID will be set to blank
  - Within the model code, checking if Z##_CID is blank or not is used as the criteria to determine if a signature exists for a given key
- Since the model is responsible for maintaining the signatures, the model developer is responsible for ensuring that the model code sets the correct CID as well.

Signatures

Signature Definition File

- Signature definitions are created by the model developer and placed to the ‘Signature Definition’ file
  - Naming convention: Z##_<CONTENT_ID_VERSION> where,
    - ## indicates the segment number and 00 <= ## <= 15



Structure of a Signature Definition File

+ This is strictly not the meaning of ‘Entity’. However, except in very rare cases, it corresponds to the segment number.

Signatures

Signature Definition File

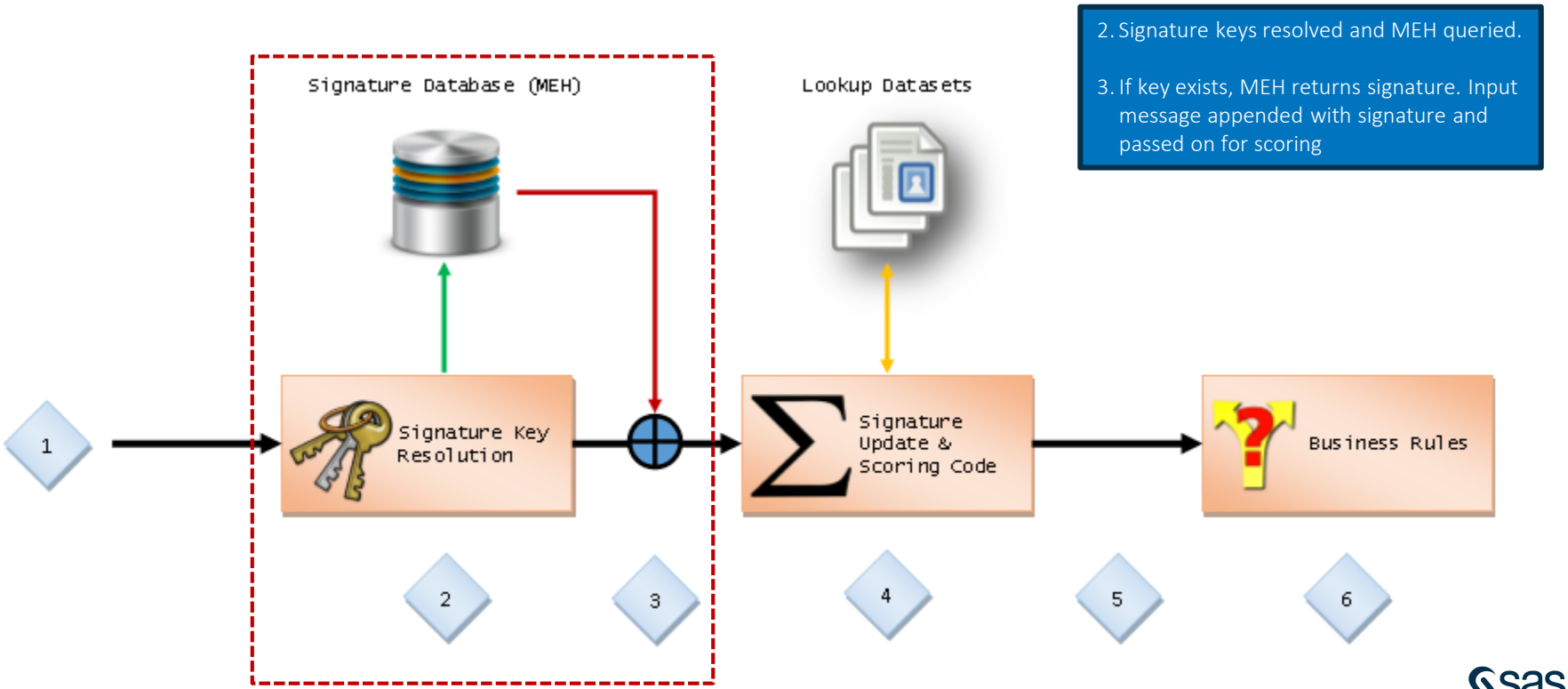
- The signature definition file is included as part of the model catalog
  - Used by the system in parsing the binary blob when reading and writing signatures to the MEH
- Each signature variable must be prefixed by Z##_ where ## indicates the segment number.
- The following order is recommended for the `length` statements:
  - Numeric arrays
  - Numeric scalars
  - Character arrays
  - Character scalars

Signatures

Fetching Signatures

Introduction to Fraud Modeling in SAS FM

RECAP: High Level Process Flow



Signatures

Fetching Signatures

- First step in fetching the signature is specifying the lookup keys
- The variable that needs to hold the lookup key for a given signature segment is Z##_lookup_key where ## indicates the segment number.

Lookup Key Resolution Process

Signatures

Fetching Signatures

- Fixed keys should be avoided as much as possible.
- There are always likely to be error / scope conditions where a lookup can be avoided.
- Resolve macros can be coded to effectively avoid such lookups.

Signatures

Z##_CID_RESOLVE Macro

- If a signature segment is specified to have a resolved key in the definition file, then it is must be accompanied by the necessary resolve logic.
- Resolve logic for each segment is written as a separate macro or file+:
  - Naming convention: %Z##_<CONTENT_ID_VERSION>_RESOLVE
    - ## indicates the segment number and 00 <= ## <= 15

```
%macro Z##_<CID>_RESOLVE / store secure;

    /* Good practice to initialize the key first */
    call missing(z##_lookup_key);

    /* Logic to derive z##_lookup_key */

%mend;
```

store required when compiling the macro into a macro catalog. secure required if the compiled code should not be exposed to the customer.
These options apply to all macros that are part of the model package.
+ Code contained within the macro can be just provided as a file to be included within the SAS datastep also (applies to all modules)

Signatures

Z##_CID_RESOLVE Examples

```
%macro Z00_CARD_EG_0100_RESOLVE / store secure;

    call missing(z00_lookup_key);
    /* Lookup key only if an authorization */
    if smh_activity_type = "CA" and not missing(hqo_card_num) then do;
        z00_lookup_key = hqo_card_num;
    end;

%mend;
```

```
%macro Z01_BENEACCT0100_RESOLVE / store secure;

    call missing(z01_lookup_key);

    /* Beneficiary account key formed by concatenating receiving bank and receiving account numbers */
    if smh_activity_type in ("SH", "BF") then do;

        /* A hypothetical macro that standardizes bank numbers in various formats (ABA, SWIFT) to a standard code */
        %standardize_bank_num(tpp_bank_num, tpp_bank_name, std_bank_num);

        if not missing(std_bank_num) and not missing(tpp_acct_num) then do;
            z01_lookup_key = std_bank_num || "-" || tpp_acct_num;
        end;

    end;

%mend;
```

Signatures

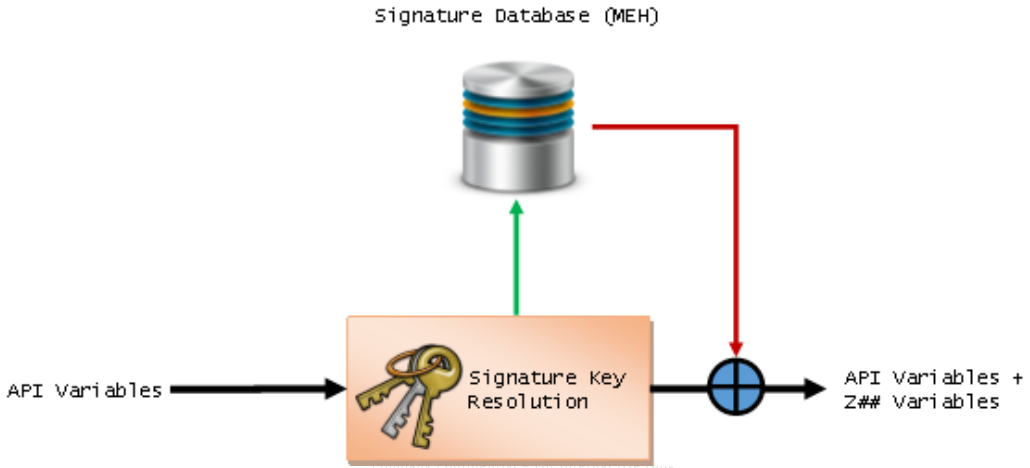
Fetching Signatures

- The system then proceeds to lookup the signature from the MEH
- All lookup keys are resolved and all segments are fetched at the same time
- This is one of the more time consuming process in the entire flow
  - Number of segments will have a direct impact on throughput performance
- Resolve macros can be used to tightly control and avoid unnecessary lookups
  - For e.g. setting the KEY to HQO_CARD_NUM in the definitions file will result in a lookup regardless of whether the transaction needs a lookup or not.
  - The sample ***%Z00_CARD_EG_0100_RESOLVE*** can eliminate this issue by looking up the signature only on required transactions

Signatures

Fetching Signatures

- If successful, the signature contents are returned
  - The signature contents are parsed back to its component variables
  - Z##_CONTENT_ID_VERSION will have the value corresponding to the signature version that was fetched
  - Incoming message is appended with the Z##_ variables



Signatures

Fetching Signatures

- Signature fetch may not be successful for various reasons:
  - Key does not exist in the MEH
    - Very common since new entities are introduced all the time
    - Z##_CONTENT_ID_VERSION as well as all other Z##_ variables will be set to missing and appended to the incoming message
  - System failures due to timeouts etc.
    - System will handle these situations.
    - Transaction will simply not enter the next steps in such cases

Signatures

RESOLVE_RECODE Macro

- In most cases, some preprocessing may be required prior to resolving the signature keys.
- For e.g., there may be some general error conditions in the transaction that prevents the transaction from getting scored.
  - In such cases, it will be efficient to not set any of the lookup keys to avoid unnecessary lookups
- Any preprocessing codes that need to execute prior to key resolution are placed within the RESOLVE_RECODE macro.
- Is an optional file

Signatures

RESOLVE_RECODE Example

```
%macro MDL_CARDMDL_0100_RESOLVE_RECODE / store secure;

&MDL._update_signature = 'Y';

/* Missing date and time - usually model does not process these transactions */

if missing(rqo_tran_date) or missing(rqo_tran_time) then do;
  &MDL._fetch_signature = 'N';
end;

/* Missing amount */

else if smh_activity_type = "CA" and missing(tca_mod_amt) then do;
  &MDL._fetch_signature = 'N';
end;

%mend;
```

Signatures

Z##_<CID>_RESOLVE that Relies on RESOLVE_RECODE Processing

```
%macro Z00_CARD_EG_0100_RESOLVE / store secure;

    call missing(z00_lookup_key);

    /* Lookup key only if an authorization */

    if smh_activity_type = "CA" and not missing(hqo_card_num)
           and &MDL._fetch_signature = 'Y' then do;
        z00_lookup_key = hqo_card_num;
    end;

%mend;
```

Signatures

Initializing and Updating Signatures

Signatures

Z##_<CID>_UPDATE Macro

- Update logic for each signature segment is written as a separate macro:
  - Naming convention: %Z##_<CONTENT_ID_VERSION>_UPDATE
    - ## indicates the segment number and 00 <= ## <= 15

```
%macro Z##_<CID>_UPDATE / store secure;

    /* Initialize signatures if necessary */

    /* Update signatures */

%mend;
```

Signatures

Initializing Signatures

- All signatures need to be initialized before first use
  - Trying to access uninitialized values can lead to run-time errors
- Whether the signature exists for a given key or not can be tested by looking at the CID value

```
if missing(Z##_CONTENT_ID_VERSION) then do;

    /* Initialize Signatures */

end;
```

Signatures

Initializing Signatures - Example

```
if missing(Z00_CONTENT_ID_VERSION) then do;

  /* Initialize bookkeeping variables */
  z00_arr_curlength = 0;
  z00_arr_curr_index = 5;
  call missing(z00_arr_prev_index);

  /* Initialize content variables */
  call missing(of z00_datetimes_:);
  .
  .
  .

  /* Set the content ID version */
  z00_content_id_version = "CARD_EG_0100";

end;
```

```
/* A necessary SAS construct to group the 10 individual elements into an array object */
/* Not just part of initialization; need to invoke every time a signature is read */
/* If not grouped as an array, array operations cannot be performed later */

array z00_datetimes_{5};
.
.
array z00_poss_{5};
```

Signatures

Inserting Signatures

- If a signature is initialized and created for the first time, then a new record will be inserted in the MEH
- Signature inserts are more expensive than signature updates
  - Some performance issues are to be expected when a large number of new signatures are created in the system. Such instances include:
    - First model
    - Introduction of new signature segment
    - Addition of a large new portfolio to the system
- This issue can be mitigated by creating an initialized empty signature and inserting them in bulk into the MEH during low traffic periods
  - Relies on creating a custom message that is used solely for initializing the signatures

Signatures

Inserting Blank Signatures in Bulk

```
if missing(Z00_CONTENT_ID_VERSION)
  /* A card master message that is used to initialize a new card signature */
  /* If this message is inadvertently sent after the signature is created, it will reset
     the signature. You can use additional logic to prevent that from happening */
  or (smh_activity_type = "MC" and tmc_init_flag = 'Y') then do;

  /* Initialize bookkeeping variables */
  z00_arr_curlength = 0;
  z00_arr_curr_index = 5;
  call missing(z00_arr_prev_index);

  /* Initialize content variables */
  call missing(of z00_datetimes_:);
  .
  .
  .

  /* Set the content ID version */
  z00_content_id_version = "CARD_EG_0100";

end;
```

Signatures

Bookkeeping Variables

- In addition to the numerous variables that contain information about the entity, the signature may also require some variables for bookkeeping purposes.
- Any number of bookkeeping variables can also be part of the signature
  - Should be kept to a minimum in order to have maximum space allocated for the actual content variables.

Signatures

Bookkeeping Variables Example: Tracking Chronological Order

Index	Date / Time	Amount	MCC	Merchant	Zip Code	POS
1	.	.	.	.	.	.
2	.	.	.	.	.	.
3	.	.	.	.	.	.
4	.	.	.	.	.	.
5	.	.	.	.	.	.

curr_index

CurLength	0

prev_index

Signatures

Bookkeeping Variables Example: Tracking Chronological Order

curr_index 

Index	Date / Time	Amount	MCC	Merchant	Zip Code	POS
1	20140301 09:27:10	77.49	5542	MARATHON #1567	92137	02
2	.	.	.	.	.	.
3	.	.	.	.	.	.
4	.	.	.	.	.	.
5	.	.	.	.	.	.

CurLength	1

prev_index

Signatures

Bookkeeping Variables Example: Tracking Chronological Order

prev_index → 1

curr_index → 2

Index	Date / Time	Amount	MCC	Merchant	Zip Code	POS
1	20140301 09:27:10	77.49	5542	MARATHON #1567	92137	02
2	20140302 13:23:01	102.08	5713	BANANA INTERNATIONAL PLA	92122	90
3	.	.	.	.	.	.
4	.	.	.	.	.	.
5	.	.	.	.	.	.

CurLength	2

Signatures

Bookkeeping Variables Example: Tracking Chronological Order

prev_index →

curr_index →

Index	Date / Time	Amount	MCC	Merchant	Zip Code	POS
1	20140301 09:27:10	77.49	5542	MARATHON #1567	92137	02
2	20140302 13:23:01	102.08	5713	BANANA INTERNATIONAL PLA	92122	90
3	20140303 12:51:35	40.54	5713	OLD NAVY-ON DRCT ONLNE	94105	01
4	.	.	.	.	.	.
5	.	.	.	.	.	.

CurLength	3

Signatures

Bookkeeping Variables Example: Tracking Chronological Order

	Index	Date / Time	Amount	MCC	Merchant	Zip Code	POS
	1	20140301 09:27:10	77.49	5542	MARATHON #1567	92137	02
	2	20140302 13:23:01	102.08	5713	BANANA INTERNATIONAL PLA	92122	90
prev_index →	3	20140303 12:51:35	40.54	5713	OLD NAVY-ON DRCT ONLNE	94105	01
curr_index →	4	20140304 18:12:55	45.22	5542	CHEVRON UTC	92122	90
	5	.	.	.	.	.	.

CurLength	4

Signatures

Bookkeeping Variables Example: Tracking Chronological Order

	Index	Date / Time		Amount	MCC	Merchant	Zip Code	POS
	1	20140301	09:27:10	77.49	5542	MARATHON #1567	92137	02
	2	20140302	13:23:01	102.08	5713	BANANA INTERNATIONAL PLA	92122	90
	3	20140303	12:51:35	40.54	5713	OLD NAVY-ON DRCT ONLNE	94105	01
prev_index →	4	20140304	18:12:55	45.22	5542	CHEVRON UTC	92122	90
curr_index →	5	20140304	18:46:20	31.19	5651	ANTHROPOLOGIE #569	92122	90

CurLength	5

Signatures

Bookkeeping Variables Example: Tracking Chronological Order

curr_index →

prev_index →

Index	Date / Time	Amount	MCC	Merchant	Zip Code	POS
1	20140304 19:21:43	42.12	5712	THE LAND OF NOD 158	92122	90
2	20140302 13:23:01	102.08	5713	BANANA INTERNATIONAL PLA	92122	90
3	20140303 12:51:35	40.54	5713	OLD NAVY-ON DRCT ONLNE	94105	01
4	20140304 18:12:55	45.22	5542	CHEVRON UTC	92122	90
5	20140304 18:46:20	31.19	5651	ANTHROPOLOGIE #569	92122	90

CurLength	5

Signatures

Bookkeeping Variables Example: Tracking Chronological Order

prev_index →

curr_index →

Index	Date / Time	Amount	MCC	Merchant	Zip Code	POS
1	20140304 19:21:43	42.12	5712	THE LAND OF NOD 158	92122	90
2	20140304 19:38:25	2.75	5814	CHAMPAGNE BAKERY 4207	92122	05
3	20140303 12:51:35	40.54	5713	OLD NAVY-ON DRCT ONLNE	94105	01
4	20140304 18:12:55	45.22	5542	CHEVRON UTC	92122	90
5	20140304 18:46:20	31.19	5651	ANTHROPOLOGIE #569	92122	90

CurLength	5

Signatures

Bookkeeping Variables Example: Tracking Chronological Order

```
/* Update number of elements in the array */
z00_arr_curlength = min(z00_arr_curlength + 1, 5);

/* Which element in the arrays contains the element from the previous transaction */
if z00_arr_curlength > 1 then z00_arr_prev_index = z00_arr_curr_index;

/* Pointer to the next open slot in the array that will hold the values from this
   transaction. If arrays are full, then the oldest element will be overwritten with values
   from this transaction */

if z00_arr_curr_index + 1 > 5 then z00_arr_curr_index = 1;
else z00_arr_curr_index + 1;
```

Chronological order here refers to the order in which transactions arrive in the system.
The actual transaction date and time may not be in chronological order. We will revisit this topic later.

Signatures

Recoding Variables

- Some preprocessing may be required on variables prior to persisting them in signatures. E.g.

```
/* Recode MCC */

length &MDL._mcc_recoded $4;

if missing(hct_mer_mcc) then call missing &MDL._mcc_recoded);
else if notdigit(strip(hct_mer_mcc)) then &MDL._mcc_recoded = "IVLD";
else &MDL._mcc_recoded = put(input(hct_mer_mcc,4.),z4.);
```

```
/* Recode merchant to keep only alpha-numeric characters */

length &MDL._merchant_recoded $40;

&MDL._merchant_recoded = compress(hct_term_owner_name, 'kad');
```

Signatures

Updating Signatures

- Final step is to push values from the current transaction into the signature

```
z00_datetimes_{z00_arr_curr_index} = rqo_tran_date * 86400 + rqo_tran_time;
z00_amounts_{z00_arr_curr_index} = tca_mod_amt;
z00_mccs_{z00_arr_curr_index} = &MDL._mcc_recoded;
z00_poss_{z00_arr_curr_index} = ucm_pos;
z00_merchants_{z00_arr_curr_index} = &MDL._merchant_recoded;

if hct_term_cntry_code = '840' then do;
  %validate_US_zip_code(hct_term_post_code, valid);
  if valid then z00_zipcodes_{z00_arr_curr_index} = hct_term_post_code;
  else z00_zipcodes_{z00_arr_curr_index} = "00000";
end;
else z00_zipcodes_{z00_arr_curr_index} = "FORGN";

&MDL._atm_amount_sum = 0;
if &MDL._mcc_recoded = "6011" then do;
  &MDL._atm_amount_sum = z00_average_atm_amount * z00_count_atm_txns + tca_mod_amt;
  z00_count_atm_txns = z00_count_atm_txns + 1;
  z00_average_atm_amount = &MDL._atm_amount_sum / z00_count_atm_txns;
end;

z00_persist_ind = 'Y';
```

Signatures

Z##_PERSIST_IND

- Z##_PERSIST_IND is a special variable that is used to indicate if the updated signature should be written back to the MEH or not after processing.
  - Signatures need not always be persisted back to the MEH
  - Certain transactions may need to read the signature for scoring, but may not update the signature
    - E.g. execution of scheduled payment, advice messages
  - Certain error / abnormal conditions detected during the signature update step may lead to not updating the signature
    - E.g. duplicate transaction detected based on previous entry in the signature
  - In some cases only a subset of segments may require an update. In such cases the other segments need not be persisted.
    - E.g. beneficiary signature may not be updated for a self-account transfer; only the account signature is updated
- This variable is not part of the signature; just an indicator variable used by the system during processing
- Valid values are ‘Y’ / ‘N’
- Good practice to initialize it to ‘N’ and set it to ‘Y’ after all the update codes
- Should be set to ‘Y’ when performing a bulk signature insert also.

Signatures

Updating Signatures

- Typically all elements of the signature are not updated by all transactions

Variable	Update Criteria
z00_datetimes z00_amount z00_mccs z00_zipcodes z00_merchants z00_pos_entry_modes	Card authorizations
z00_average_atm_amount z00_count_atm_txns	Card authorizations at an ATM terminal
z00_last_address_change_dt	Address change non-monetary / masterfile updates
z00_credit_limit	Credit limit change non-monetary / masterfile updates

Signatures

Updating Signatures

- Arrays that are updated based on different update criterion should be grouped separately and have their own bookkeeping.

```
length z00_datetimes_1 - z00_datetimes_5                    8.;
length z00_amount_1 - z00_amounts_10                        8.;
length z00_mccs_1 - z00_mccs_5                              $4.;
length z00_zipcodes_1 - z00_zipcodes_5                      $5.;
length z00_merchants_1 - z00_merchants_5                    $40.;
length z00_pos_entry_modes_1 - z00_pos_entry_modes_5        $2.;

length z00_arr_curlength                                    8.;
length z00_arr_curr_index                                   8.;
length z00_arr_prev_index                                   8.;
```

Set of arrays and corresponding bookkeeping variables for authorizations

```
length z00_be_datetimes_1 - z00_be_datetimes_3              8.;
length z00_terminals_1 - z00_terminals_3                    $40.;

length z00_be_arr_curlength                                 8.;
length z00_be_arr_curr_index                                8.;
length z00_be_arr_prev_index                                8.;
```

Set of arrays and corresponding bookkeeping variables for balance enquiries

- Arrays can be sized differently depending on frequency of updating events
- Almost always the attributes being stored will be different; storing as one group will waste a lot of space.
- Allows much faster and efficient processing later during feature derivation

Signatures

Updating Signatures

Type	Date / Time	Amount	MCC	Merchant	Zip Code	POS	Terminal
CA	20140304 19:21:43	42.12	5712	THE LAND OF NOD 158	92122	90	
CA	20140304 19:38:25	2.75	5814	CHAMPAGNE BAKERY 4207	92122	05	
CA	20140303 12:51:35	40.54	5713	OLD NAVY-ON DRCT ONLNE	94105	01	
NM	20140304 18:12:55						BANK12342365
CA	20140304 18:46:20	31.19	5651	ANTHROPOLOGIE #569	92122	90	
NM	20140304 18:12:55						BTR23423
NM	20140304 18:46:20						ST34245-234

Date / Time	Amount	MCC	Merchant	Zip Code	POS
20140304 19:21:43	42.12	5712	THE LAND OF NOD 158	92122	90
20140304 19:38:25	2.75	5814	CHAMPAGNE BAKERY 4207	92122	05
20140303 12:51:35	40.54	5713	OLD NAVY-ON DRCT ONLNE	94105	01
20140304 18:46:20	31.19	5651	ANTHROPOLOGIE #569	92122	90

Date / Time	Terminal
20140304 18:12:55	BANK12342365
20140304 18:12:55	BTR23423
20140304 18:46:20	ST34245-234

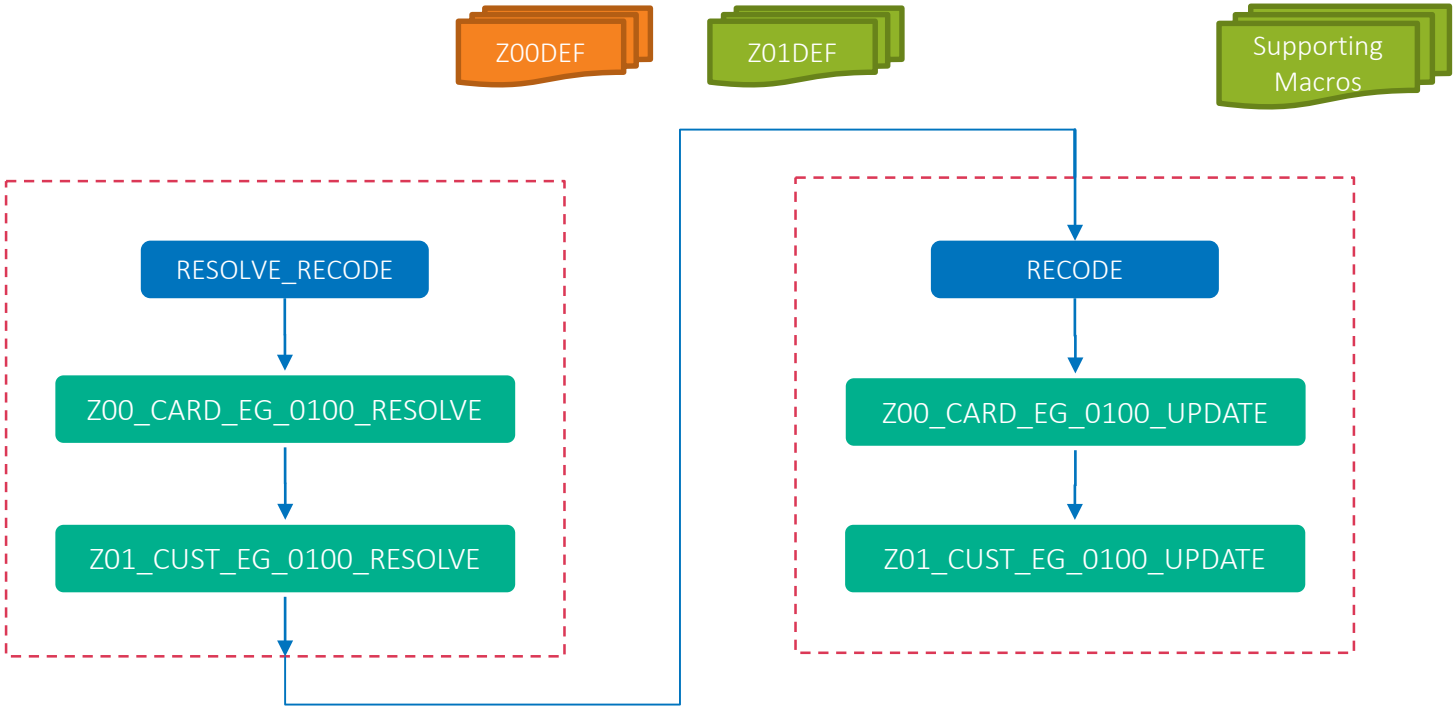
Signatures
RECODE Macro

- Recoding steps can be part of the Z##_<CID>_UPDATE or the RESOLVE macros
- If the preprocessing step applies to multiple segments, then it is more efficient to place it in the RECODE macro.
  - E.g. model uses an customer and card signature, both of which have an MCC array; in such cases the MCC recode is required by both signature updates
- Steps prior to and after fetching signatures will run on two different threads
  - Temporary variables computed in the RESOLVE_RECODE and RESOLVE macros will not be made available to the UPDATE and SCORE macros.
  - Therefore any error checks performed in prior steps may have to be repeated again
    - Refer to “RESOLVE_RECODE Example” slide.
  - Those codes can be just re-executed in the RESOLVE macro
- RECODE is an optional macro

Signatures

Putting it All Together

- With an additional customer signature



Signatures

Applications

Signatures
Applications

- Feature generation
- Blacklisting / whitelisting entities
- Fraud feedback at entity or transaction level
- House-holding
- Network analysis

```
/* Feature generation: Average transaction amount during the last 24 hours */

call missing(&MDL._feature_average_amount_in_24_hours);

&MDL._this_time = z00_datetimes_{z00_arr_curr_index};
&MDL._sum_spend_24_hours = 0;
&MDL._count_24_hours = 0;

/* Reverse chronologically traverse array */

&MDL._curr_pointer = z00_arr_curr_index;
&MDL._prev_pointer = z00_arr_prev_index;

/* Loop through the arrays starting from the most current index to the oldest index */

do ii = 1 to z00_arr_curlength;

  /* Is this transaction within the past 24 hours? */
  if &MDL._this_time - z00_datetimes_{&MDL._curr_pointer} <= 86400 then do;
    &MDL._sum_spend_24_hours = &MDL._sum_spend_24_hours + z00_amount_{&MDL._curr_pointer};
    &MDL._count_24_hours = &MDL._count_24_hours + 1;
  end;

  /* Update pointers for next iteration */

  if not missing(&MDL._prev_pointer) then do;
    if &MDL._prev_pointer - 1 < 1 then &MDL._prev_pointer = z00_arr_curlength;
    else &MDL._prev_pointer  = &MDL._prev_pointer - 1;
  end;

  if &MDL._curr_pointer - 1 < 1 then &MDL._curr_pointer = z00_arr_curlength;
  else &MDL._curr_pointer  = &MDL._curr_pointer - 1;

end;

&MDL._feature_average_amount_in_24_hours = &MDL._sum_spend_24_hours / &MDL._count_24_hours;
```

```
/* Feature Generation: Percentage of spend at this MCC */

call missing(&MDL._feature_percentage_of_spend_at_mcc);

/* Reverse chronologically traverse array */

&MDL._curr_pointer = z00_arr_curr_index;
&MDL._prev_pointer = z00_arr_prev_index;

&MDL._this_mcc = z00_mcc_{z00_arr_curr_index};
&MDL._sum_spend_this_mcc = 0;
&MDL._sum_spend_all_mcc  = 0;

do ii = 1 to z00_arr_curlength;

  /* Spend at this MCC */
  if z00_mcc_{&MDL._curr_pointer} = &MDL._this_mcc then &MDL._sum_spend_this_mcc = &MDL._sum_spend_this_mcc + z00_amount_{&MDL._curr_pointer};
  /* Total spend */
  &MDL._sum_spend_all_mcc = &MDL._sum_spend_all_mcc + z00_amount_{&MDL._curr_pointer};

  /* Update pointers for next iteration */

  if not missing(&MDL._prev_pointer) then do;
    if &MDL._prev_pointer - 1 < 1 then &MDL._prev_pointer = z00_arr_curlength;
    else &MDL._prev_pointer  = &MDL._prev_pointer - 1;
  end;

  if &MDL._curr_pointer - 1 < 1 then &MDL._curr_pointer = z00_arr_curlength;
  else &MDL._curr_pointer  = &MDL._curr_pointer - 1;

end;


&MDL._feature_percentage_of_spend_at_mcc = &MDL._sum_spend_this_mcc / &MDL._sum_spend_all_mcc * 100;
```

Signatures

Blacklisting / Whitelisting

- A scalar variable in the signature can be updated to hold this information

```
/* Possible values are B (blacklist), W (whitelist), N (none) */
z00_black_white_list = "N";
```

- Send a custom non-monetary message to indicate that an entity is whitelisted or blacklisted

```
/* z00_black_white_list set to 'N' during signature initialization */
if tng_tran_type = "LL" and tng_sub_tran_type = "WL" then z00_black_white_list = "W";
else if tng_tran_type = "LL" and tng_sub_tran_type = "BL" then z00_black_white_list = "B";
```

- Model can utilize this information.
  - Can use it in scoring
  - Or stamp the transaction to be used by rule-writers (eliminates lookup lists)

Signatures

Fraud Feedback

- Fraud feedback at entity or transaction level
  - Entity level feedback is very similar to white/black listing; just an indicator to indicator if the entity has / had fraud on it
- Tracking transaction level fraud requires an array of fraud indicators
- Send a custom message to indicate that a particular transaction is fraud
  - May contain pertinent information corresponding to the fraud transaction such that the it can be located within the signature via a fuzzy search
  - Alternately may contain a unique transaction identifier
    - This requires that every transactions arrives with a unique transaction identifier and is stored in the signature arrays
  - Not guaranteed to find the transaction since it may have been flushed out of the signature
- Again the fraud indicators can be heavily utilized by the model for scoring

Signatures

Design Considerations

Signatures

Design Considerations: Choice of Signature Key

- Signature entry for a given lookup key is locked while the transaction is being processed
- Requires special consideration for batch processing where numerous transactions for a given lookup key can arrive simultaneously for processing.
  - E.g. large number of nightly deposit submissions from a single account
  - May need some analytic or system workarounds:
    - Experimenting with updating at regular intervals as opposed to updating on every transaction and studying the impact on the models
    - Sorting the batch prior to entering SAS FM to remove such large continuous blocks
    - Use the orchestration layer to append a ‘moving’ summary of the batch to the incoming message
- This also poses a constraint on the entities on which a signature can be keyed
  - Entities with low cardinality can be very problematic. E.g. country codes, MCCs etc.

Signatures

Design Considerations: Choosing Elements

- What elements gets stored in the signature is a key design consideration
- Two approaches:
  - Top – down
    - Start with as many data elements as possible in the signature based on the transaction
    - Derive features based on the 'extended' signature
    - Pick the elements based on the final set of selected features
    - Essentially a form of feature selection
  - Bottom – up
    - Select elements based on domain knowledge and / or intuition on which elements are needed
- In both cases, all signature elements that we start with will not be required to drive the final set of features
  - Extraneous elements should be removed

Signatures

Design Considerations: Sizing Arrays

- The number of events to persist in the signature will have a direct impact on the analytic performance of the model
- No systematic method (unless you try a brute force method of building many iterations with different array sizes)
- Typical approaches include:
  - Based on the distributions of transactions per entity per month.
    - E.g. say the 95th percentile is 20 transactions per months for a card portfolio. This means that an array of size 120 will hold 6 months of history for 95% of the cardholders.
  - Based on (maximum allowed size – size of all scalar variables) / (size of one row)
    - Gives the maximum possible array size
    - No room for future upgrades. Can factor in a margin in the above calculation
  - Domain knowledge
    - You just know that about N events for a certain channel is sufficient to give a good behavioral profile

Signatures

Design Considerations: Sizing Individual Elements

- Some space savings can be achieved by carefully choosing the size of individual elements
- Not all numeric fields need to be stored as 8 byte floats; depending on the context, they can be anywhere between 3 and 8 bytes.
- Some character elements can consume a lot of space
  - E.g. merchant name is 40 bytes long. An array of 50 elements can take up about 2kB space.
  - Can come up with ways to compress them; truncation, hashing etc.
- Carefully avoiding redundancies
  - When you build very large signatures, redundancies can inherently creep in
  - E.g. you don't to store an array of indicators that indicate if the transaction is foreign if you already have an array of terminal country codes

Signatures

Signature Priming

- Signature priming is the process where the signature elements start getting populated
- Priming period can be a period of ‘high entropy’ in the system:
  - Model scores and its analytic performance can fluctuate rapidly
    - Since models are trained using a well primed signature
  - System performance can also be impacted
    - High number of record inserts into the MEH can be a bottleneck
- Signature priming starts when a new model goes into production
  - There have been various attempts at priming offline and moving the primed signatures into production
  - Those are complex processes that is out of scope for our discussions

Signatures

Signature Priming

- When is the signature ‘fully primed’?
  - There is no fixed criteria for this
  - Can be measured in terms of:
    - Time it takes for the score distributions (or any other model output) to stabilize (PSI measures)
    - Time it takes for the model performance to reach steady state performance
- Typical priming periods range from 2 week to 8 weeks.
  - Models that have entities that transact quickly tend to prime quickly
    - E.g. cards
  - Models that have entities that transact sparsely tend to prime slowly
    - E.g. payments
  - Different sub-portfolios can prime at different rates also
    - E.g. commercial payment accounts prime quickly relative to personal payment accounts
  - Function of signature design as well
    - Large arrays will take longer to prime

Signatures

Upgrading and Downgrading Signatures

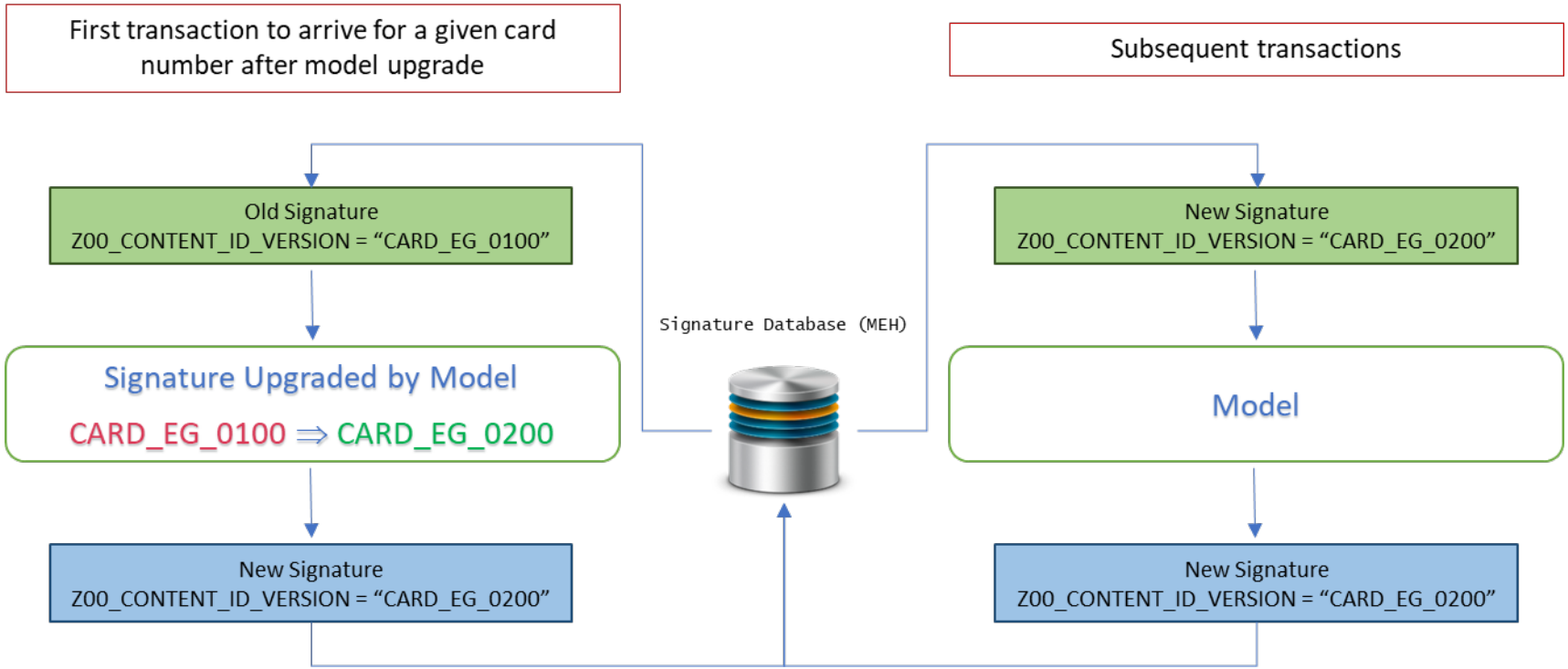
Signatures

Upgrading Signatures

- When a model is rebuilt / refreshed, it is very likely that:
  - new elements are added
  - existing elements are removed
- Changing the signature from an existing structure to a new structure is termed as a ‘signature upgrade’
- When upgrading a signature:
  - The signature should be assigned a new CID
  - The new model should understand both the old and new signature versions
- You can upgrade / downgrade only within a single family of signatures

Signatures

Signature Upgrade Process



Signatures

Signature Upgrade Example

- Consider the following upgraded signature where:
  - A new array is added
  - A new scalar is added

```
length z00_datetimes_1 - z00_datetimes_5              8.;
length z00_amount_1 - z00_amounts_5                   8.;

length z00_last_address_change_dt                     8.;
length z00_credit_limit                               8.;
length z00_average_atm_amount                         8.;
length z00_count_atm_txns                             8.;
length z00_arr_curlength                              8.;
length z00_arr_curr_index                             8.;
length z00_arr_prev_index                             8.;

length z00_mccs_1 - z00_mccs_5                        $4.;
length z00_zipcodes_1 - z00_zipcodes_5                $5.;
length z00_merchants_1 - z00_merchants_5              $40.;
length z00_poss_1 - z00_poss_5                        $2.;
length z00_cntrys_1 - z00_cntrys_5                    $3.;

length z00_been_abroad_before                         $1.;
```

Signatures

Method 1 for Upgrading Signatures: Reset Entire Signature

```
/* If CID anything other than CARD_EG_0200 reset the entire signature */

if missing(Z00_CONTENT_ID_VERSION) or Z00_CONTENT_ID_VERSION ~= "CARD_EG_0200" then do;

  /* Initialize bookkeeping variables */
  z00_arr_curlength = 0;
  z00_arr_curr_index = 5;
  call missing(z00_arr_prev_index);

  /* Initialize content variables */
  call missing(of z00_datetimes_:);
  call missing(of z00_amount_:);
  call missing(of z00_mccs_:);
  call missing(of z00_zipcodes_:);
  call missing(of z00_merchants_:);
  call missing(of z00_poss_:);
  call missing(of z00_cntrys_:);
  call missing(z00_last_address_change_dt);
  call missing(z00_credit_limit);
  call missing(z00_average_atm_amount);
  call missing(z00_count_atm_txns);
  call missing(z00_been_abroad_before);

  /* Set the content ID version */
  z00_content_id_version = "CARD_EG_0200";

end;
```

Signatures

Method 1 for Upgrading Signatures: Reset Entire Signature

- Most simple method
  - Models need not deal with prior signature versions
  - No additional logic other than checking for the required CID
- Practically problematic
  - Entire signature needs to be primed from scratch when a new model goes in
  - A very hard sell for most customers

Signatures

Method 2 for Upgrading Signatures: Incremental Initialization

```
if missing(Z00_CONTENT_ID_VERSION) then do;

  /* Initialize bookkeeping variables */
  z00_arr_curlength = 0;
  z00_arr_curr_index = 5;
  call missing(z00_arr_prev_index);

  /* Initialize content variables */
  call missing(of z00_datetimes_:);
  call missing(of z00_amount_:);
  call missing(of z00_mccs_:);
  call missing(of z00_zipcodes_:);
  call missing(of z00_merchants_:);
  call missing(of z00_poss_:);
  call missing(of z00_cntrys_:);
  call missing(z00_last_address_change_dt);
  call missing(z00_credit_limit);
  call missing(z00_average_atm_amount);
  call missing(z00_count_atm_txns);
  call missing(z00_been abroad_before);

  /* Set the content ID version */
  z00_content_id_version = "CARD_EG_0200";

end;
```

```
else if Z00_CONTENT_ID_VERSION = "CARD_EG_0100" then do;

  /* Initialize only the incremental part */
  call missing(of z00_cntrys_:);
  call missing(z00_been abroad_before);

  /* Set the content ID version */
  z00_content_id_version = "CARD_EG_0200";

end;
```

Signatures

Method 2 for Upgrading Signatures: Incremental Initialization

- More complex migration codes
- Requires that the differences between signature versions are tracked carefully and are handled separately within the code.
  - More than 1 previous CID may exist in the MEH
    - In these situations, you could combine the two approaches
    - Incremental initialization of signature versions that are only up to a few versions old
    - Completely reset the signature for much older versions
- Practically more appealing
  - Only the incremental part needs to be primed
  - Priming periods are usually much less than priming an entire signature

Signatures

Removing Elements During Upgrade

- Consider the following upgraded signature where:
  - A new array and a new scalar are added
  - An array and 2 scalars are removed

```
length z00_datetimes_1 - z00_datetimes_5                8.;
length z00_amount_1 - z00_amounts_5                     8.;

length z00_last_address_change_dt                       8.;
length z00_credit_limit                                 8.;
length z00_average_atm_amount                           8.;
length z00_count_atm_txns                               8.;
length z00_arr_curlength                                8.;
length z00_arr_curr_index                               8.;
length z00_arr_prev_index                               8.;

length z00_mccs_1 - z00_mccs_5                          $4.;
length z00_zipcodes_1 - z00_zipcodes_5                  $5.;
length z00_merchants_1 - z00_merchants_5                $40.;
length z00_poss_1 - z00_poss_5                          $2.;
length z00_cntrys_1 - z00_cntrys_5                      $3.;

length z00_been_abroad_before                           $1.;
```

Signatures

Method 2 for Upgrading Signatures: Incremental Initialization

```
if missing(Z00_CONTENT_ID_VERSION) then do;

  /* Initialize bookkeeping variables */
  z00_arr_curlength = 0;
  z00_arr_curr_index = 5;
  call missing(z00_arr_prev_index);

  /* Initialize content variables */
  call missing(of z00_datetimes_:);
  call missing(of z00_amount_:);
  call missing(of z00_mccs_:);
  call missing(of z00_zipcodes_:);
  call missing(of z00_merchants_:);
  call missing(of z00_poss_:);
  call missing(of z00_cntrys_:);
  call missing(z00_last_address_change_dt);
  call missing(z00_credit_limit);
  call missing(z00_average_atm_amount);
  call missing(z00_count_atm_txns);
  call missing(z00_been abroad_before);

  /* Set the content ID version */
  z00_content_id_version = "CARD_EG_0200";

end;
```

```
else if Z00_CONTENT_ID_VERSION == "CARD_EG_0100" then do;

  /* Initialize only the incremental part */
  call missing(of z00_cntrys_:);
  call missing(z00_been abroad_before);

  /* Set the content ID version */
  z00_content_id_version = "CARD_EG_0200";

end;
```

- No additional steps are required to remove the extraneous elements during an upgrade
  - The system will simply ignore those variables when writing it back to the MEH
- However removing elements have other more complex considerations …..

Signatures

Downgrading Signatures

- Occasionally, after a rebuilt model is deployed, it may have to be pulled out and replaced with the previous model for various reasons:
  - Performance issues (technical and analytical)
  - Certain dependencies not met for the new model to work properly
    - Key additional messages not provisioned yet
  - Improper rule calibration leading to very high / low alert volumes
    - To buy more time and redo the calibration analysis
- However some signatures may have upgraded to the new CID during the time the new model was executing
- Therefore after downgrading the model, the old model will likely see the new CID
  - Old model should be able to understand a future CID

Signatures

Downgrading Signatures

- Typically models are built to understand three CIDs:
  - Previous CID for an upgrade situation
  - Future CID for downgrade situation
  - Current CID
- All other CIDs are just simply reset
  - Not an absolute requirement, but makes signature management simple
    - The model developer can choose to support as many previous versions as they want as long as they handle each situation correctly within the code
  - More plausible to get customer on board on this limitation than a full reset of everything

Signatures

Supporting 3 CIDs

```
/* If CID anything other than current, previous or one future versions, reset the entire signature */

if missing(Z00_CONTENT_ID_VERSION) or
  Z00_CONTENT_ID_VERSION not in ("CARD_EG_0100", "CARD_EG_0000", "CARD_EG_0200") then do;

  /* Initialize bookkeeping variables */
  z00_arr_curlength = 0;
  z00_arr_curr_index = 5;
  call missing(z00_arr_prev_index);

  /* Initialize content variables */
  call missing(of z00_datetimes_:);
  call missing(of z00_amount_:);
  call missing(of z00_mccs_:);
  call missing(of z00_zipcodes_:);
  call missing(of z00_merchants_:);
  call missing(of z00_poss_:);
  call missing(z00_last_address_change_dt);
  call missing(z00_credit_limit);
  call missing(z00_average_atm_amount);
  call missing(z00_count_atm_txns);
  /* Set the content ID version */
  z00_content_id_version = "CARD_EG_0100";

end;
```

Signatures

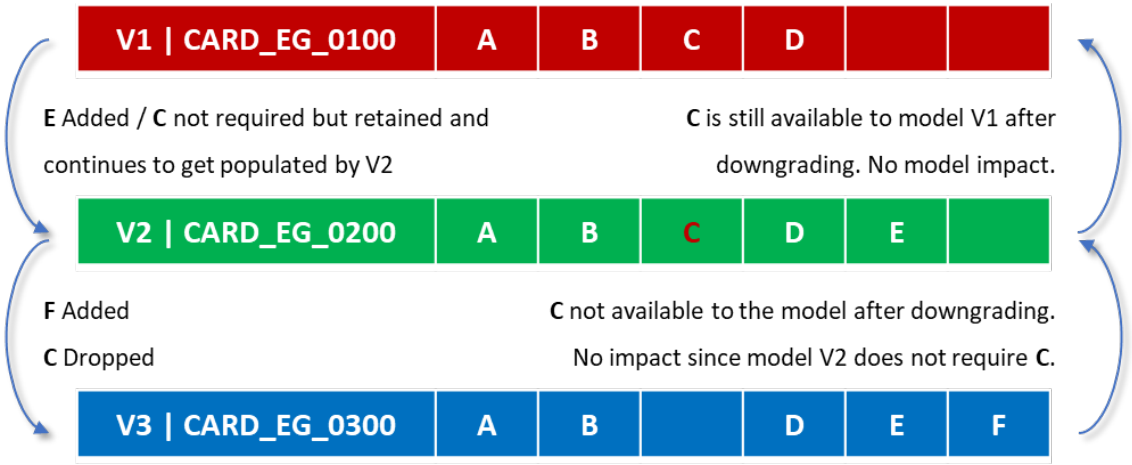
Downgrading From Future Version

- When downgrading from a future version, no additional coding is required on the newly added elements
  - The current CID does not know the existence of those elements and therefore will simply be ignored
- However if the future CID had removed any elements that exists in the current CID, it may cause issues
  - They will be set to missing since they were not part of the incoming future CID
    - This is due to SAS behavior (not SAS FM)
    - The variables exist in the data step PDV, but not set to any value in the incoming message
  - It is not possible to initialize them appropriately since this model cannot have knowledge about the future state of the signature

Signatures

Downgrading From Future Version

- The failproof way to avoid this situation is to not drop any elements between two consecutive versions if it is used by the previous version
  - i.e. continue to support elements that were part of the last version (V1) despite not being required by the next version (V2)
  - Those elements are dropped from the signature on a future version (V3)



Signatures

Downgrading From Future Version

- This also ensures that the previous model can continue to operate as if it was never pulled out
  - Operationally highly beneficial since the customer can start at where they left
  - This is the whole point of downgrading to a previous model

Signatures

More Signature Constructs

Signatures

More Signature Constructs

- With a bit of creativity, various complex data structures can be constructed within the signatures
- We will look at two other constructs:
  - Visit summaries
  - Trees

Signatures

Visit Summaries

- Used to hold a longer summary of past visits / interactions with other entities.
  - Contains slowly changing aggregate information unlike event based signature arrays
  - E.g. Country visit summary contains aggregate information about the cardholders' visits to other countries
    - Each country has just one entry associated with it; individual events are not tracked
  - Constructed using a combination of arrays and scalars along with its own bookkeeping variables

Signatures

Example: Country Visit Summary

Country Key	First Visit	Last Visit	Number of Visits	Number of Transactions	Amount Average	Amount STD
124	__	__	__	__	__	__
392	__	__	__	__	__	__
144	__	__	__	__	__	__
826	__	__	__	__	__	__
036	__	__	__	__	__	__

CurLength

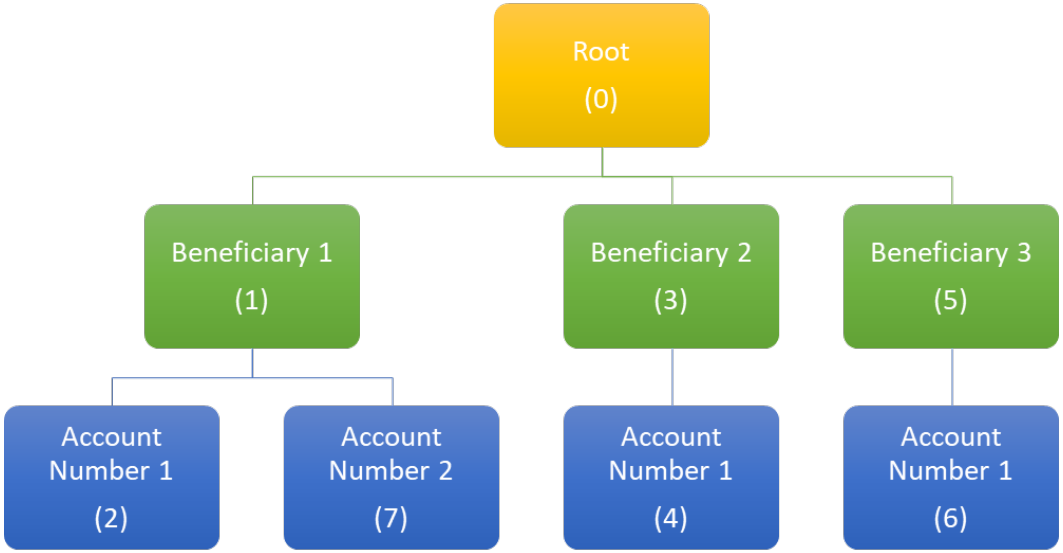
Multiple arrays in ‘sync’ via their indices as before. Key array contains unique country codes.

When space runs out, a FIFO or FILO strategy can be used to replace entries.
Alternately other criteria such as number of visits or transactions can be used to determine the least important entry for removal

Signatures

Trees

- Consider the following tree you want to maintain for the most frequent beneficiaries (identified by a standardized name) to whom money has been transferred to:



Signatures

Example: Beneficiary – Account Tree

Entity	Parent	Avg. Amount	# Accounts
Root	-	---	4
B1	0	---	2
A11	1	---	x
B2	0	---	1
B21	3	---	x
B3	0	---	1
A31	5	---	x
A12	1	---	x

Nodes

Beneficiaries

Signatures

The End