Configuration of Functions

# Entity Types, Data Spaces, Key Spaces from ABC Applications, and Member Subscriptions

1. Define the application. Define the ENTITY TYPES for the application in the (new) table APPLICATION\_ENTITY\_TYPE. This is the set of conceptual objects of the application. The same entity types may be used by multiple applications, hence this table.
2. For each ABC Application, define a parent Data Space (type of “ABC APPLICATION”).
3. Record a Member’s subscription to an ABC Application. For each subscription to the application, define a Bridging Context DATA\_SPACE (type of “ABC SUBSPACE”) as a child context to the ABC APPLICATION data space..
4. For every ENTITY TYPE associated with the Application, define a KEY\_SPACE to act as the roll-up of member-specific key spaces.
5. When a member subscribes, in addition to the data space (subspace) derived, also derive KEY\_SPACEs by making one child Key Space for each Application, Entity Type, and Subscription, make these each roll up to the appropriate parent KEY\_SPACE (Application and Entity Type).
6. Next, for every KEY\_SPACE derived (both parents and children), generate one KEY\_SPACE\_ID\_FUNCTION\_TMPL and one ABSTRACT\_FUNCTION of type [ID]. Thus one [ID] template function will be derived for every APPLICATION, ENTITY TYPE and SUBSCRIPTION.
7. Using the Insight UX, create an [ID] FUNCTION clone of the correct ABSTRACT\_FUNCTION for each OSF\_FILE\_TYPE that contains columns which satisfy the definition of the Abstract Function Slots. Add an OSF\_ID\_FUNCTION entry to tie the new FUNCTION to the OSF\_FILE\_TYPE. This should also create required BINDING\_TERMS tying the OSF\_FILE\_TYPE columns to the instance of the cloned FUNCTION\_SLOTS.

NOTE: One unique OSF\_FILE\_TYPE has the primary key: SMART\_OSF\_ID, CFG\_VERSION\_ID. Hence at least one function per combination.

# [ID] Functions

Special functions representing the identity of an instance of some object/entity. Every Key Space defines one and only one [ID] function.

[ID] functions have any number of named input parameter slots, corresponding number of named output slots each distinctly related to the input slots, plus two additional output slots. The “KEY VALUE” is the concatenation of the keyspace columns; and “\_ENTITY\_KEY\_” is the derived, Insight, surrogate key value, which is unique across all data spaces. The KEY VALUE represents the unique value within the context.

Each input slot is paired automatically with an output slot with a similar name. The function portion of the [ID] function merely passes the input slot out to the similarly named output slot. The key value is derived through concatenation of the input slot values in the sequence indicated by the slot number and uses a standard delimiter (currently the tilde “~”).

In actuality, the Key Match service calculates a unique Entity Key Id for any unique combination of parameter values in order to provide a convenient database identifier for the concept. This is done actually by applying the Entity Key Id Sequence Generator Function to each unassigned, unique key value combinations.

Hence, in some respects, the [ID] function isn’t really a function in the sense that its inputs are calculated upon to create its outputs.

It has input slots to which other elements can be bound, and it provides output slots that can themselves be bound to other things.

The function is always a specific INSTANCE of the function. Hence, while there is an abstract concept of an [ID] function defining the set of columns defining the ID within a specific Key Space, each unique usage of such a function (i.e., each unique BINDING of the function) must be captured as its own entry in the database. If a particular Key Space’s [ID] function must be derived for the Key Match process, and separately for some mapping transformation,

## Configuration of [ID] Function Meta Data

1. Define a unique Key Space for an Entity Type and Data Space. An Entity Type may have multiple [ID] functions so long as they each are defined within a different Context (Data Space and Key Space). Each Key Space should represent a unique combination of Entity Attributes and physical representations defined by the Data Space.
2. Construct the abstract [ID] function name by appending the name of the ENTITY\_TYPE to the conceptual attribute “[ID]”. Associate one abstract [ID] function to each Key Space defined for the Entity Type.
3. In the ABSTRACT FUNCTION table use the constructed function name from step 2. The [ID] function is invertible between the Key Value and its parameters. This functionality may not be implemented however.
4. The FUNCTOR\_SLOTS defines one standard OUTPUT column, named “KEY\_VALUE”. This could be bound to an alias name representing the key value, although there’s no functionality that uses this today. Create a corresponding KEY\_VALUE OUTPUT slot in the ABSTRACT\_FUNCTION\_SLOTS template table.
5. The FUNCTOR\_SLOTS defines a second, standard OUTPUT column, named “\_ENTITY\_KEY\_ID\_”. This is a Data Gateway reserved word. The [ID] function will fill this column with the database-assigned ENTITY\_KEY\_ID value from the ENTITY\_KEY table. [[1]](#footnote-1)
6. Define a set of (at least one) Attributes/Column names representing the logical parts of the Key Space ID. These should be named simply and as generically as possible. For example, a single column ID for a “Student” entity type might be named “STUDENT\_ID”. A multi-column ID for a “Taxpayer” entity might include “SSN” and “COUNTRY”.
7. For each logical key column, create both INPUT and OUTPUT slots. The OUTPUT slots will use the column names directly as defined in the previous step. The INPUT slots should be constructed by appending suffixes “\_IN” to the OUTPUT slot names. Store all column names in the ABSTRACT\_FUNCTION\_SLOTS.
8. SUMMARY: The ABSTRACT FUNCTION\_SLOTS should include all INPUT and OUTPUT Slots for each specific [ID] function. The OUTPUT Slot would be tied to the FUNCTOR\_SLOTS entry for the Key Value. The INPUT slots and other OUTPUT slots would not appear in the FUNCTOR\_SLOTS table, as they are unspecified until defined here. For an [ID] function, the INPUT SLOTS are the logical names of the input columns. For example, the input slot “STUDENT ID\_IN” might be the singleton parameter for the [STUDENT].[ID] key function. Multi-part [ID]s in particular must define and name all input slots and the order in which they appear in the key.
9. Then for each use of the ABSTRACT FUNCTION, the full abstraction should be copied and assigned a new FUNCTION\_ID value. Once this is generated, bind this instance’s INPUT slots to appropriate other entries. For example, to use a copy of the ID function in the Key Match service, tie an instance to the SOURCE\_QUERY by adding an entry to the SOURCE\_QUERY\_FUNCTION table, and bind the function’s INPUT slots to appropriate QUERY\_COLUMN entries associated to that SOURCE\_QUERY.
10. To use the function, all of its INPUT slots must be bound using BINDING\_TERM entries. Each function must be associated with at least one process. These could be one of the Standard Processes, such as Key Match, Population Membership or Fulfillment, or it could potentially be some future, non-standard process as well.

## Binding Terms

### Subspace Partition Built-In

When the data space is a SUBSPACE within an application context, then the Member SK field on the child DATA\_SPACE record will be filled with the MBR\_SK value of the Application Subscriber. This acts as the MARKER for the SUBSPACE when it is represented in the parent Data Space (in other words, it is the Data Space Partition marker symbol), and may be referenced within the BINDING\_TERM as the "SUBSPACE\_PARTITION" built-in value. Other markers may be used in the future to designate Data Space subspaces, which would require some modifications here.

## Standard Processes: Key Match Service

There are two known Key Match Services defined to create and maintain Entity Ids using [ID] Functions. The first is the Proof Of Concept Harvester Key Match Service which processes the results of queries executed by the Harvester components. The second is the OSF Archive Key Match Service which processes the received OSF files submitted by members using SMART. Additional services could be developed as additional types of ID indexing applications are defined.

### Harvester Key Match Service

Uses the following tables to trigger the service and locate the query results files:

SOURCE\_QUERY

QUERY\_RUN

SOURCE\_QUERY\_ID\_FUNCTION (renamed to HARVESTER\_ID\_FUNCTION)

QUERY\_COLUMN

COLUMN\_IN\_QUERY

RECORD\_STRUCTURE

STRUCTURE\_COLUMN

This service fills the MATCHED\_KEY (renamed HARVESTER\_MATCHED\_KEY), in addition to the standard ENTITY\_KEY and ENTITY\_KEY\_COLUMN tables.

### OSF Archive Key Match Service

Uses the following tables to trigger the service and to locate the OSF file to be indexed:

OSF\_ID\_FUNCTION

OSF\_FILE

OSF\_RUN

RECORD\_STRUCTURE

STRUCTURE\_COLUMN

This service fills the OSF\_MATCHED\_KEY, in addition to the standard ENTITY\_KEY and ENTITY\_KEY\_COLUMN tables.

# CHOOSER Functions

A simple function applied to all rows of a dataset and returning a single column value per row. Given an arbitrary list of input slots bound to various dataset column names, and a preference ordering on these slot names, the function returns the first NON-NULL column value it finds on the row. Variations of the function include the ability to choose to keep any value from one of the slots, even if NULL, if all prior columns in the preference order were also NULL. If all columns are NULL, then the value is set to NULL.

The Simple Chooser acts exactly as described, but returns a single value tied to a specific Attribute. It is used by the Fulfillment engine to select the candidate Attribute value when more than one possible origin exists for an attribute.

The Chooser takes any number of INPUT SLOTS, undefined by the FUNCTOR, defined by the FUNCTION instance.

## Configuration of Simple Chooser Function Meta Data

1. For each Attribute in the Attribute Set of a Population, create a “Simple Chooser” function whose name is the Attribute Name.
2. For each Attribute Binding to a Column in a Query, create an INPUT slot whose Alias is constructed from the Attribute Name and the SOURCE QUERY ID where the column is found. If the column appears in multiple queries, then create multiple entries, one for each query.
3. Put these Slot Aliases in the CHOOSER\_FUNCTION\_BINDING table (for now) as INPUT slots. Set the SLOT NUMBER value to indicate the preferred order of choice.
4. If there is a default query from which even a NULL value should take precedence, set the “CHOOSE\_IF\_NULL” flag for that slot.
5. Define the OUTPUT Slot Alias by inserting a row into FUNCTION\_BINDING where the Slot Number 1 is given the Slot Alias name to be used as the output value. For these Attribute chooser functions, the CHOICE slot should be given an alias which is the same as the ATTRIBUTE NAME. Binding requires a physical reference, so at the moment use any correct structure and column name, even if that is not the column that gets chosen.

NOTE: As of today, OUTPUT slots must be defined and BOUND to a physical column using the FUNCTION\_BINDING table. The current data model has not allowed for defining unbound slot aliases, which is what we should have for these Functions. The database and code will need to be altered to

TO DO: Change Chooser Function Slot Aliasing to search for INPUT SLOTS not OUTPUT SLOTS

# Population Constraints

Population Constraints are special kinds of functions. They are implied functions over a set of constraints, which themselves are Boolean functions. Constraints are either TRUE or FALSE, and only TRUE constraints result in the inclusion of a key in a population.

Configuration is slightly different. The entries in the POPULATION\_CONSTRAINTS table must be AND-ed together to render the correct value for a row. These constraints must be defined in such a manner as only their TRUE state is valid.

For example, say a Population needs members who satisfy one constraint but which do not satisfy another. Then in configuration and implementation, there should be two constraints defined:

<CONSTRAINT 1>

<NOT CONSTRAINT 2>

There is currently no “NOT” operator on constraints, so defining a positive constraint to indicate the negative condition is the only way to accomplish the required logic.

Each POPULATION\_CONSTRAINT entry will be turned into a Boolean function, whose function name and output slot alias share the constraint name. These repetitions of the constraint name only occur in memory and are not explicitly stored.

POPULATION\_CONSTRAINT entries cannot be referenced directly by other generic functions and therefore cannot be bound to other functions. They are, however, bound implicitly by the POPULATION MATCH SERVICE to a Boolean conjunction function (namely “F\_AND”) where they are bound as input parameter slots. A population candidate member is accepted into the population if all of its POPULATION\_CONSTRAINT entries are determined to be TRUE.

To configure the Population Constraint functions, the following tables most have data:

POPULATION

POPULATION\_CONSTRAINT

POPULATION\_CONSTRAINT\_BINDING

BINDING\_TERM

Plus any others referenced through the BINDING\_TERM entries.

1. This value is generated by one of the Key Match algorithms using an embedded Cached Sequence Generator Function. All other uses of the [ID] function will simply use the cached values to fill this column. Optionally, the [ID] Function can be configured to not fill this column, [↑](#footnote-ref-1)