model

DICTIONARY [V -> attached ANY, K -> attached ANY]

feature
model: FUN [K, V]

ensure
consistent_model_imp_counts: model.count = count
consistent_model_imp_contents: ∀x : 1 ≤ j ≤ Result.count :
Result.has (create {PAIR [K, V]}.make (keys [j.item], values [j.item]))

feature {NONE}
values: ARRAY[V]
keys: LINKED_LIST[K]

feature -- Commands
add_entry (v: V; k: K)
require
non_existing_in_model: not model.domain.has (k)
ensure
entry_added_to_model: model ~ old model.extended
(create {PAIR[K,V]}.make_from_tuple ([k, v]))
remove_entry (k: K)
require
existing_in_model: model.domain.has (k)
ensure
entry_removed_from_model: model ~ (old model.deep_twin.domain_subtracted_by (k))

feature -- Constructor

make -- Initialize an empty dictionary.
ensure
empty_model: model.is_empty
object_equality_for_keys: keys.object_comparison
object_equality_for_values: values.object_comparison

feature -- Queries

count: INTEGER_32
-- Number of keys in BST.
ensure
correct_model_result: model.count = count

get_keys (v: V): ITERABLE [K]
-- Keys that are associated with value 'v'.
ensure
correct_model_result: ∀x : 1 ≤ j ≤ Result.count :
model.range_restricted_by (v).domain.has (j.item)
end

get_value (k: K): detachable V
-- Assocated value of 'k' if it exists.
-- Void if 'k' does not exist.
ensure
case_of_void_result: not model.domain.has (k) ⇒ (Result = Void)
case_of_non_void_result: model.domain.has (k) ⇒ (not (Result = Void))

feature -- feature required by ITERABLE
new_cursor: ITERATION_CURSOR [TUPLE [V, K]]

invariant
consistent_keys_values_counts: keys.count = values.count
consistent_imp_adt_counts: keys.count = count

\+
TUPLE_ITERATION_CURSOR

tests

INSTRUCTOR_DICTIONARY_TESTS

root

TEST_DICTIONARY