Team members	Name	Prism Login	Signature
Member 1:	Gefei Song		
Member 2:			
*Submitted under Prism account:			

Contents:

Documentation must be done to professional standards. See OOSC2 Chapter 26: *A sense of style*. Code and contracts must be documented using the Eiffel and BON style guidelines and conventions. *CamelCase* is used in Java. In Eiffel the convention is *under_score*. Attention must be paid to using appropriate names for classes and features. Class names must be upper case, while features are lower case. Comments and header clauses are important. For class diagrams, use the BON conventions, and use clusters as appropriate. Use the EiffelStudio document generation facility (e.g. text, short, flat etc. RTF views), suitably edited and indented to prevent wrapping, to help you obtain appropriately documentation (e.g. contract views). Each diagram must be at the appropriate level of abstraction. Use Visio for the BON class diagrams.

Your signature attests that this is your own work and that you have obeyed university academic honesty policies. Academic honesty is essentially giving credit where credit is due, and not misrepresenting what you have done and what work you have produced. When a piece of work is submitted by a student it is expected that all unquoted and uncited ideas and text are original to the student. Uncited and unquoted text, diagrams, etc., which are not original to the student, and which the student presents as their own work is considered academically dishonest.

1. Requirements for tracker project

Our customer provided us with the following statement of their needs: A tracker system monitors the position of waste products in nuclear plants and ensures their safe handling. Our customer requires a software system that operators use to manage safe tracking of radioactive waste in their various nuclear plants.

We have so far elicited the following information from our customer. Containers of material pass through various stages of processing in the tracking part of the nuclear plant. The tracking plant consist of several phases usually corresponding to the physical processes that handle the radioactive materials. Not all plants have precisely the same phases.

As an example, containers (containing a possibly radioactive material type) might arrive at an initial unpacking phase where they are stored for further processing depending on their material contents. All nuclear plants have only the following types of material: glass, metal, plastic, liquid. No other materials are tracked.

A subsequent phase might be called the assay phase to measure the recoverable material content of each container before passing onto the next phase. A next stage might be a compacting phase. A compacting phase might involve dissolving metal contents or crushing glass. Not all material types can necessarily be handled in a phase. For example, we should not move containers with liquid into a compacting phase. Finally, the products of the process might be placed in storage. There may be other phases in an instance of the tracker.

Each container has a unique identifier and contains only one type of material. It is labelled with a preliminary radiation count (in mSv). When a container is registered in the system, it is also placed in a phase (not necessarily an initial phase).

The sievert (symbol: Sv) is a unit of ionizing radiation dose in the International System of Units (SI) and is a measure of the health effect of radiation on the human body. Quantitiesthat are measured in sieverts are intended to represent the stochastic health risk, which for radiation dose assessment is defined as the probability of cancer induction and genetic damage. One sievert carries with it a 5.5% chance of eventually developing cancer.1

For a given plant, there is an initial setup of two important fixed global parameters: there is a limit on the maximum radiation in any phase of the plant (in units of mSv), and there is also a limit on the maximum radiation that any container in the plant may have (in mSv). An error status message shall be signaled if there is an attempt to register a new container in the system with radiation that exceeds the container limit.

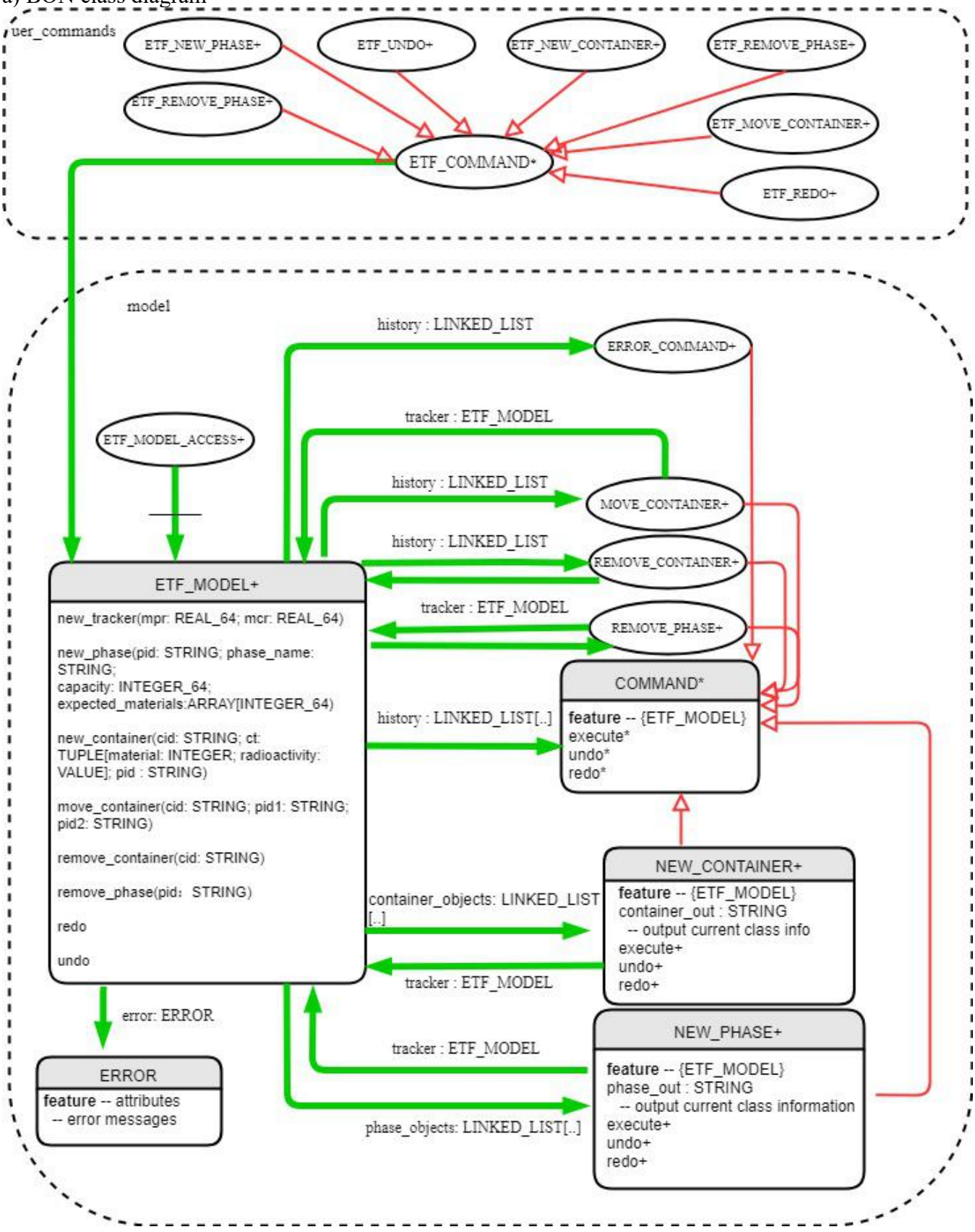
Another operation is to add a new phase (this is information provided by the Domain experts). Requirements elicitation so far yields that a new phase is specified by a phase ID, a name (e.g. "compacting"), a limit on the maximum number of containers in the phase, and a list of material types that may be treated in the phase. A phase may also be removed if there are no containers anywhere in the system. Also, it is possible for an operator to move a container from one phase to another.

Obviously when dealing with dangerous materials, it is very important to ensure that no material goes missing and that care is taken to avoid too much radioactive material getting into a phase, in case there is a buildup of dangerous substances in one area. The tracking manager is responsible for giving permission to movements of containers between processing phases to avoid dangerous situations.

See *tracker.definitions.txt* in the appendix for the grammar of the user interface. The acceptance tests *at1.expected.txt, at2.expected.txt* and *at3.expected.txt* describe some of the input-output behavior at the console for this project.

2. BON class diagram overview (*architecture* of the design)

a) BON class diagram



b) Overall design and Main design decisions

Overall design:

In this system, user can input eight different commands : new_tracker(..), new_phase(..), new_container(..), move_container(..), remove_container(..), remove_phase(..), redo and undo. Input commands can be received by the ETF_COMMAND_INTERFACE, and distribute to its subclass, such as ETF_NEW_PHASE, ETF_NEW_CONTAINER, etc.), errors are determined by those corresponding class, and call set_message to pass those error messages to the ETF_MODEL class and the ETF_MODEL class will call ETF_CMD_LINE_OUTPUT_HANDLER to output these errors to the command line. All successfully executed commands(except redo, undo and new_tracker) will be store in a linked_list called "history"(in ETF_MODEL class) for redo and undo mechanism.

Main design decisions:

Because the system should support redo/undo mechanism, so i choose command pattern to build our system. In the model cluster, i defined a deferred class called COMMAND with with four deferred features: execute, redo, undo and is_error. We also have NEW_PHASE, NEW_CONTAINER, MOVE_CONTAINER, REMOVE_PHASE, REMOVE_CONTAINER, ERROR_COMMAND classes, those classes will inherit from COMMAND class and override these four features. Command pattern will encapsulate each command as a single class. Once user input a valid command from command line, the system will pass this command to the ETF_MODEL class. In the ETF_MODEL class, we also have features like new_phase, new_container, ect, those features have the same name as the input commands and each feature has its corresponding class. When ETF_MODEL receive the command, its will create an instance from one of the command classes (from NEW_PHASE, NEW_CONTAINER ...), and call associated "execute" (based on dynamic binding and polymorphism) method to run the command. Also, after executing the command, store the command object into history list redo and undo.

For example:
If user input command: new_phase(..) , and the system will call the corresponding method in ETF_MODEL class which is also called new_phase(..) ,and then implement this method. At this time this method will create an object of class NEW_PHASE, this object will call an command: execute, to really implement user’s instruction. Finally store this object into a linked_list(history : LINKED_LIST[COMMAND]), based on the dynamic binding and polymorphism, users can call different versions of execute, redo, undo methods. Also by moving cursor, which is point to history list, back and forth, users can implement redo and undo mechanism.

3. Table of modules - responsibilities and information hiding

1	ETF_MODEL	Responsibility: an tracker system main class	Alternative: none
	Concrete	Secret: none	

1.1	HISTORY[COMMAND]	Responsibility: store successfully executed command objects	Alternative: See LINKED_LIST[G]
	Concrete	Secret: ordered by executed time	

1.2	PHASE_OBJECTS[NEW_PHASE]	Responsibility: store class NEW_PHASE objects.	Alternative: see LINKED_LIST[G]
	Concrete	Secret: unordered list of NEW_PHASE objects	

1.3	CONTAINER_OBJECTS[NEW_CONTAINER]	Responsibility: store NEW_CONTAINER class objects	Alternative: see LINKED_LIST[G]
	Concrete	Secret: unordered collection of NEW_CONTAINER	

1.4	SORTED_PHASE[NEW_PHASE]	Responsibility: store exact same objects in PHASE_OBJECTS but ordered by pid	Alternative: see SORTED_TWO_WAY_LIST[G]
	Concrete	Secret: none	

1.5	SORTED_CONTAINER[NEW_CONTAINER]	Responsibility: store exact same objects in CONTAINER_OBJECTS but ordered by cid	Alternative: see SORTED_TWO_WAY_LIST[G]
	Concrete	Secret: none	

2	COMMAND	Responsibility: each user command has a corresponding class, all inherit from COMMAND with four deferred features	Alternative: none
	Deferred	Secret: Dynamic binding and polymorphism	

2.1	NEW_PHASE	Responsibility: store phase data	Alternative: none
	Concrete	Secret: see COMMAND	

2.2	NEW_CONTAINER	Responsibility: store container data and features inherited from COMMAND	Alternative: none
	Concrete	Secret: see COMMAND	

2.3	MOVE_CONTAINER	Responsibility: move one container from one phase to another and change two phases' data	Alternative: none
	Concrete	Secret: see COMMAND	

2.4	REMOVER_CONTAINER	Responsibility: remove container from phase	Alternative: none
	Concrete	Secret: see COMMAND	

2.5	REMOVE_PHASE	Responsibility: remove phase from tracker if no container in it	Alternative: none
	Concrete	Secret: see COMMAND	

2.6	ERROR_COMMAND	Responsibility: set input error information and create error object	Alternative: none
	Concrete	Secret: see COMMAND	

3	ERROR	Responsibility: store all the error messages	Alternative: none
	Concrete	Secret: no command and query, just string attributes	

4. Expanded description of design decisions

In ETF_MODEL class, it is the main client class in tracker program which contains features such as new_tracker(..), new_phase(..), new_container(..), move_container(..), remove_container(..), remove_phase(..), redo and undo. It can access eight supplier classes: one deferred class COMMAND, and its six children classes: NEW_PHASE, NEW_CONTAINER, MOVE_CONTAINER, REMOVE_PHASE, REMOVE_CONTAINER, ERROR_COMMAND, and
a ERROR class which includes all the possible error messages. In ETF_MODEL class, we have six linked_list which called: history: LINKED_LIST[COMMAND], phase_objects: LINKED_LIST[NEW_PHASE], container_objects: LINKED_LIST[NEW_CONTAINER], sorted_phase: LINKED_LIST[NEW_PHASE], and sorted_container: LINKED_LIST[NEW_CONTAINER].

Every time user call a command, in ETF_MODEL class will create an corresponding object from one of those six children classes, and save its objects reference into history list, users can redo and undo executed commands by moving list's cursor back and forth to point to different command objects, and based on this object's dynamic type to determine which version of redo and undo will be called.

Every time user call new_phase(..), and new_container(..) methods, we should also save its object's reference into phase_objects, or container_objects list. These two list record different phases and containers information ordered by executed time. Due to the output on the command line must be sorted by pid and cid, so there are two commands(sort_phase_objects and sort_container_objects) in ETF_MODEL, we can sort phase_objects and container_objects lists and save their objects in the sorted_phase and sorted_container lists. There is an redefined method called "out" which can output phase and container's information on command line based on sorted list.

5. Significant Contracts (Correctness)

ETF_MODEL +

phase_objects : LINKED_LIST[NEW_PHASE]
sorted_phase : LINKED_LIST[NEW_PHASE]
 --sort linked_list phase_objects by pid
container_objects: LINKED_LIST[NEW_CONTAINER]
sorted_container: LINKED_LIST[NEW_CONTAINER]
 --sort linked_list phase_contianer by cid
max_phase_radiation: REAL_64
max_container_radiation : REAL_64
phase_of(pid : STRING) : NEW_PHASE
 -- given a pid and choose its corresponding object from phase_objects and return
container_of(cid : STRING) : NEW_CONTAINER
 -- given a cid and choose its corresponding object from container_objects and return
history: LINKED_LIST[COMMAND]
 -- store executed command for undo and redo

new_tracker(mpr : REAL_64; mcr : REAL_64)
--create a new tracker with max_phase_radiation and max_container_radiation
? mpr > 0 and mcr >0 and mpr >= mcr
! max_phase_radiation = mpr and max_container_radiation = mcr

new_phase(pid : STRING; phase_name: STRING; capacity: INTEGER_64; expected_material : LINKED_LIST[INTEGER_64])
--create a NEW_PHASE object and store it in phase_objects
? pid.at(1)∈ A-Z, a-z or 0..9 and phase_name.at(1)∈ A-Z, a-z or 0..9 and capacity > 0 and
¬ expected_materials.is_empty
! phase_objects.count = old phase_objects.count + 1 and
∃ x ∈ phase_objects : x.pid_new ~ pid and
∀ x ∈ old phase_objects.deep_twin : phase_objects.has(x) and
∃ x ∈ history : x ~ Current and
history.count = (old history.deep_twin).count + 1

new_container(cid : STRING; ct : TUPLE[material: INTEGER_64; radioactivity: VALUE]; pid : STRING)
--create a NEW_CONTAINER object and store it in container_objects
? cid.at(1)∈ A-Z, a-z or 0..9 and pid.at(1) ∈ A-Z, a-z or 0..9 and
∃ x ∈ phase_objects : x.pid_new ~ pid and
∃ x ∈ phase_objects : x.pid_new ~ pid → x.expected_material_new.has(ct.material) and
ct.radioactivity.as_double >= 0 and ct.radioactivity.as_double <= max_container_radiation
! container_objects.count = old container_objects.count + 1 and
∃ x ∈ container_objects : x.cidc ~ cid and
∀ x ∈ old container_objects.deep_twin : container_objects.has(x) and
∃ x ∈ history : x ~ Current and
history.count = (old history.deep_twin).count + 1

```
move_container(cid : STRING; pid1 : STRING; pid2 : STRING)
                -- move one container from pid1 to pid2
    ? ∃ x ∈ container_objects : x.cidc ~ cid and  ∃ x ∈ phase_objects : x.pid_new ~ pid1 and
       ∃ x ∈ phase_objects : x.pid_new ~ pid2 and pid1 /~ pid2 and
       container_of(cid).pidc ~ pid1
    !  container_of(cid).pidc ~ pid2 and ∃ x ∈ history : x ~ Current and
       history.count = (old history.deep_twin).count + 1

remove_container(cid : STRING)
                -- remove one container from phase
    ? ∃ x ∈ container_objects : x.cidc ~ cid
    ! ¬ container_objects.has(container_of(cid)) and
       ∃ x ∈ history : x ~ Current and
       history.count = (old history.deep_twin).count + 1

remove_phase(pid : STRING)
                -- remove phase when there is no container in it
    ? ∃ x ∈ phase_objects : x.pid_new ~ pid and phase_of(pid).count = 0
    !  ¬ phase_objects.has(phase_of(pid)) and
       ∃ x ∈ history : x ~ Current and
       history.count = (old history.deep_twin).count + 1

set_message2(set2 : STRING)
 -- set output error message and store this command into history list
     ? set2 /= void
     !  state_message /= void
                    -- state_message is the type of STRING for error output

undo
 -- undo command from history list
     !  history.count = (old history.deep_twin).count

redo
 -- redo command from history list
     !  history.count = (old history.deep_twin).count
```

6. Summary of Testing Procedures

a) Table of all acceptance tests

Test file	Description	Passed
At1.txt	set tracker radiation parameters create some phases add some containers move some containers phases are sorted by pid containers are sorted by cid	yes
At2.txt	report error messages for new tracker and phases -- comments may occur at the beginning or in the middle of a line	yes
At3.txt	set tracker radiation parameters create some phases add some containers move some containers undo and redo all those cammands	yes
At4.txt	Similar to at3.txt and report more errors based on different error inputs	yes

b) Provide a screen shot of ESpec unit tests that you ran.

TEST_SUITE

Note: * indicates a violation test case

PASSED (8 out of 8)

Case Type	Passed	Total
Violation	0	0
Boolean	8	8
All Cases	8	8

State	Contract Violation	Test Name
Test1	TEST_COMMAND	
PASSED	NONE	t1: test command new_phase precondition and postcondition
PASSED	NONE	t2: test command new_container precondition and postcondition
PASSED	NONE	t3: test command move_container precondition and postcondition
PASSED	NONE	t4: test command remove_container precondition and postcondition
PASSED	NONE	t5: test command remove_phase precondition and postcondition
PASSED	NONE	t6: test command undo postcondition
PASSED	NONE	t7: test command redo postcondition
PASSED	NONE	t8: test command new_trakcer precondition and postcondition

7. Appendix (Contract view of all classes and acceptance tests)

a) Contract view of all classes in model cluster

```
                        -- Automatic generation produced by ISE Eiffel --
note
        description: "A default business model."
        author: "Jackie Wang"
        date: "$Date$"
        revision: "$Revision$"

class interface
        ETF_MODEL

create {ETF_MODEL_ACCESS}
        make

feature -- model attributes

        error: ERROR

        state_count: INTEGER_64

        old_state_count_int: INTEGER_64

        old_state_count: STRING_8

        state_message: STRING_8

        p: STRING_8

        c: STRING_8

        max_phase_radiation: FORMAT_DOUBLE

        max_phase_radiation1: REAL_64
                        -- assign the the value of max_phase_radiation to the type of REAL_64

        max_container_radiation: FORMAT_DOUBLE

        max_container_radiation1: REAL_64
                        -- assign the value of max_container_radiation to the type of REAL_64
                        -- new_phase attributes

        phase_objects: LINKED_LIST [NEW_PHASE]
                        --      phase_objects_num : INTEGER

        sorted_phase: LINKED_LIST [NEW_PHASE]
                        --new_container attributes

        container_objects: LINKED_LIST [NEW_CONTAINER]

        container_objects_num: INTEGER_32

        sorted_container: LINKED_LIST [NEW_CONTAINER]
                        --history list store executed commands

        history: LINKED_LIST [COMMAND]

feature -- command

        sort_phase_objects
                        -- sort objects in the list "phase_objects" to increasing order accroding to
"pid" and store the sorted list into a new array which called "sorted_phase"
                ensure
                        order_changed_elements_unchanged: across
                                                sorted_phase as cursor
                                        all
                                                phase_objects.has (cursor.item)
                                        end

        sort_container_objects
                        -- sort objects in the list "container_objects" to increasing order accroding to
"cid" and store the sorted list into a new list which called "sorted_container"
```

```
		ensure
			order_changed_elements_unchanged: across
					sorted_container as cursor
				all
					container_objects.has (cursor.item)
				end

	set_message2 (set2: STRING_8)
			--set ok or error message for output
		require
			valid_input: set2 /= Void
		ensure
			valid_output: state_message /= Void

	set_message (set: STRING_8)
		ensure
			correct_state_message: state_message ~ set

	set_old_state_count (number: INTEGER_64)
			-- set old state count in undo
		require
			valid_number: number > 0
		ensure
			valid_string: old_state_count ~ "(to " + (number - 1).out + ") "

	set_old_state_count2 (number2: INTEGER_64)
			-- set old state count in redo
		require
			valid_number: number2 > 0
		ensure
			valid_string: old_state_count ~ "(to " + number2.out + ") "

	removed_one_container_objects (new_container_objects: LINKED_LIST [NEW_CONTAINER])
			--after removing one element from "conatiner_objects", store the rest of them
into "container_objects"

	removed_one_phase_objects (new_phase_objects: LINKED_LIST [NEW_PHASE])
			--after removing one element from "phase_objects", store the rest of them into
"phase_objects"

	clear_sorted_container
			--clear array which called "sorted_continer"
		ensure
			list_is_empty: sorted_container.is_empty

	clear_sorted_phase
			--clear array which called "sorted_phase"
		ensure
			list_is_empty: sorted_phase.is_empty

	default_update

	reset
			-- Reset model state.

feature -- queries

	container_of (cc: STRING_8): NEW_CONTAINER
			-- input cid and return its corresponding object from container_objects
		require
			container_not_empty: not container_objects.is_empty
			container_exist: across
					container_objects as cursor
				some
					cursor.item.cidc ~ cc
				end
		ensure
			Result.cidc ~ cc

	phase_of (phase_id: STRING_8): NEW_PHASE
			--input pid and return its corresponding object from phase_objects
		require
			phase_not_empty: not phase_objects.is_empty
			phase_exist: across
```

```
                                        phase_objects as cursor
                                some
                                        cursor.item.pid_new ~ phase_id
                                end
                ensure
                                Result.pid_new ~ phase_id

feature -- model commands

        new_tracker (mpr: REAL_64; mcr: REAL_64)
                require
                        positive_phase_radiation: mpr > 0.to_double
                        positive_container_radiation: mcr > 0.to_double
                        phase_radiation_greater_than_container_radiation: mcr <= mpr
                ensure
                        max_phase_radiation1_not_void: max_phase_radiation1 = mpr
                        max_container_radiation1_not_void: max_container_radiation1 = mcr
                        state_message_ok: state_message ~ "ok"

        new_phase (pid: STRING_8; phase_name: STRING_8; capacity: INTEGER_64; expected_materials: ARRAY
[INTEGER_64])
                require
                        valid_pid: pid.at (1).is_alpha_numeric
                        valid_phase_name: phase_name.at (1).is_alpha_numeric
                        valid_capacity: capacity > 0
                        exit_epected_materials: not expected_materials.is_empty
                ensure
                        phase_objects_count_increased: phase_objects.count = (old
phase_objects.deep_twin).count + 1
                        phase_objects_added: across
                                        phase_objects as cursor
                                some
                                        cursor.item.pid_new ~ pid
                                end
                        other_phase_objects_unchanged: across
                                        old phase_objects.deep_twin as cursor
                                all
                                        cursor.item.pid_new /~ pid
                                end
                        state_message_ok: state_message ~ "ok"
                        history_count_increased: history.count = (old history.deep_twin).count + 1

        new_container (cid: STRING_8; ct: TUPLE [material: INTEGER_64; radioactivity: VALUE]; pid:
STRING_8)
                        --create new container
                require
                        valid_cid: cid.at (1).is_alpha_numeric
                        valid_pid: pid.at (1).is_alpha_numeric
                        phase_exist: across
                                        phase_objects as c0
                                some
                                        c0.item.pid_new ~ pid
                                end
                        valid_material: across
                                        phase_objects as c1
                                some
                                        c1.item.pid_new ~ pid implies c1.item.expected_material_new.has
(ct.material)
                                end
                        valid_radioactivity: ct.radioactivity.as_double >= 0.to_double and
ct.radioactivity.as_double <= max_container_radiation1
                ensure
                        container_objects_count_increased: container_objects.count = (old
container_objects.deep_twin).count + 1
                        container_objects_num_increased: container_objects_num = old
container_objects_num + 1
                        container_objects_added: across
                                        container_objects as cursor
                                some
                                        cursor.item.cidc ~ cid
                                end
                        other_container_objects_unchanged: across
                                        old container_objects.deep_twin as cursor
                                all
```

```
                                cursor.item.cidc /~ cid
                        end
                not_greater_than_max_phase_radiation: across
                                phase_objects as c3
                        some
                                c3.item.pid_new ~ pid implies c3.item.radiation1 <=
max_phase_radiation1
                        end
                state_message_ok: state_message ~ "ok"
                history_count_increased: history.count = (old history.deep_twin).count + 1

        move_container (cid: STRING_8; pid1: STRING_8; pid2: STRING_8)
                --move one container from one phase to another
                require
                        container_exist: across
                                container_objects as cursor
                        some
                                cursor.item.cidc ~ cid
                        end
                source_phase_exist: across
                                phase_objects as cursor
                        some
                                cursor.item.pid_new ~ pid1
                        end
                target_phase_exist: across
                                phase_objects as cursor
                        some
                                cursor.item.pid_new ~ pid2
                        end
                pid_different: pid1 /~ pid2
                container_already_in_source_phase: container_of (cid).pidc ~ pid1
                ensure
                        success_moved: container_of (cid).pidc ~ pid2
                        history_count_increased: history.count = (old history.deep_twin).count + 1

        remove_container (cid: STRING_8)
                -- remove one container from phase
                require
                        container_exist: across
                                container_objects as cursor
                        some
                                cursor.item.cidc ~ cid
                        end
                ensure
                        not container_objects.has (container_of (cid))
                        history_count_increased: history.count = (old history.deep_twin).count + 1

        remove_phase (pid: STRING_8)
                --remove phase when there is no contianer in it
                require
                        phase_exist: across
                                phase_objects as cursor
                        some
                                cursor.item.pid_new ~ pid
                        end
                        phase_has_no_container: phase_of (pid).count = 0
                ensure
                        success_removed: not phase_objects.has (phase_of (pid))
                        history_count_increased: history.count = (old history.deep_twin).count + 1

        undo
                ensure
                        history.count = (old history.deep_twin).count

        redo
                ensure
                        history.count = (old history.deep_twin).count

feature -- output

        out: STRING_8
                -- New string containing terse printable representation
                -- of current object
```

```
end -- class ETF_MODEL
                        -- Generated by ISE Eiffel --
                        -- For more details: http://www.eiffel.com --

                        -- Automatic generation produced by ISE Eiffel --
note
        description: "Singleton access to the default business model."
        author: "Jackie Wang"
        date: "$Date$"
        revision: "$Revision$"

expanded class interface
        ETF_MODEL_ACCESS

create
        default_create

feature

        M: ETF_MODEL

invariant
                M = M

end -- class ETF_MODEL_ACCESS
                        -- Generated by ISE Eiffel --
                        -- For more details: http://www.eiffel.com --

note
        description: "Summary description for {NEW_PHASE}."
        author: ""
        date: "$Date$"
        revision: "$Revision$"

class interface
        NEW_PHASE

create {ETF_MODEL, NEW_PHASE, REMOVE_PHASE}
        make

feature -- attributes

        tracker: ETF_MODEL

        pid_new: STRING_8

        phase_name_new: STRING_8

        capacity_new: INTEGER_64

        count: INTEGER_32
                        --number of containers in current phase

        radiation: FORMAT_DOUBLE

        radiation1: REAL_64

        expected_material_new: ARRAY [INTEGER_64]

        materials: ARRAY [STRING_8]

        old_state_count: INTEGER_64
                        -- store the old state count

        current_index: INTEGER_32

feature -- commands

        increase_count
                ensure
                                count = old count + 1

        decrease_count
                ensure
```

```
                        count = old count - 1

        increase_radiation1 (r: REAL_64)
                ensure
                        radiation1 = old radiation1 + r

        decrease_radiation1 (r: REAL_64)
                ensure
                        radiation1 = old radiation1 - r

feature --inherited commands

        execute
                --execute current command

        redo
                -- redo current command
                ensure then
                        history_unchanged: tracker.history.count = (old tracker.history).count

        undo
                -- undo current command
                ensure then
                        history_unchanged: tracker.history.count = (old tracker.history).count

feature -- queries

        is_error: BOOLEAN
                -- no error(set to False) if we successfully create an object of current class
                ensure then
                        Result = False

        phase_out: STRING_8
                -- output current class information
                ensure
                        valid_output: Result /= Void

        remove_phase_object (o: NEW_PHASE): LINKED_LIST [NEW_PHASE]
                require
                        valid_input: o /= Void
                ensure
                        list_not_void: Result /= Void

end -- class NEW_PHASE

note
        description: "Summary description for {NEW_CONTAINER}."
        author: ""
        date: "$Date$"
        revision: "$Revision$"

class interface
        NEW_CONTAINER

create {ETF_MODEL, REMOVE_CONTAINER}
        make

feature -- attributes

        tracker: ETF_MODEL

        cidc: STRING_8

        con: TUPLE [m: INTEGER_64; rad: VALUE]

        pidc: STRING_8

        con_m: INTEGER_64

        con_r: REAL_64

        con_r_format: FORMAT_DOUBLE

        material: STRING_8
```

```
	old_state_count: INTEGER_64
			-- store the old state count

feature -- inherited commands and query

	is_error: BOOLEAN
		ensure then
				Result = False

	execute

	redo
		ensure then
			history_unchanged: tracker.history.count = (old tracker.history).count

	undo
		ensure then
			history_unchanged: tracker.history.count = (old tracker.history).count

feature -- query

	container_out: STRING_8
			-- output current class information
		ensure
			valid_output: Result /= Void

	set_pid (pid: STRING_8)
			-- set pid to current class
		require
			not_void: pid /= Void
		ensure
			valid_pid: pidc ~ pid

end -- class NEW_CONTAINER

note
	description: "Summary description for {MOVE_CONTAINER}."
	author: ""
	date: "$Date$"
	revision: "$Revision$"

class interface
	MOVE_CONTAINER

create {ETF_MODEL}
	make

feature --attributes

	tracker: ETF_MODEL

	cid: STRING_8

	pid1: STRING_8

	pid2: STRING_8

	old_state_count: INTEGER_64
			-- store the old state count

feature -- inherited commands and query

	is_error: BOOLEAN
		ensure then
				Result = False

	execute

	redo

	undo
```

```
end -- class MOVE_CONTAINER

note
	description: "Summary description for {REMOVE_CONTAINER}."
	author: ""
	date: "$Date$"
	revision: "$Revision$"

class interface
	REMOVE_CONTAINER

create
	make

feature --initialization

	make (t: ETF_MODEL; c: STRING_8)
		ensure
			valid_tracker: tracker ~ t
			valid_cid: cid ~ c

feature -- attributes

	tracker: ETF_MODEL

	cid: STRING_8

	old_state_count: INTEGER_64
			-- store the old state count

	old_container: NEW_CONTAINER

feature -- inherited commands and qury

	is_error: BOOLEAN
		ensure then
				Result = False

	execute

	redo
		ensure then
			history_unchanged: tracker.history.count = (old tracker.history).count

	undo
		ensure then
			history_unchanged: tracker.history.count = (old tracker.history).count

feature -- query

	remove_container_object (o: NEW_CONTAINER): LINKED_LIST [NEW_CONTAINER]
		require
			valid_input: o /= Void
		ensure
			vlaid_output: Result /= Void

end -- class REMOVE_CONTAINER

note
	description: "Summary description for {REMOVE_PHASE}."
	author: ""
	date: "$Date$"
	revision: "$Revision$"

class interface
	REMOVE_PHASE

create {ETF_MODEL}
	make

feature -- attributes

	tracker: ETF_MODEL
```

```
        pid: STRING_8

        old_phase_object: NEW_PHASE

        old_state_count: INTEGER_64

        current_index: INTEGER_32

feature -- inherited commands and query

        is_error: BOOLEAN
                ensure then
                        Result = False

        execute

        redo
                ensure then
                        history_unchanged: tracker.history.count = (old tracker.history).count

        undo
                ensure then
                        history_unchanged: tracker.history.count = (old tracker.history).count

feature --query

        remove_phase_object (o: NEW_PHASE): LINKED_LIST [NEW_PHASE]
                require
                        valid_input: o /= Void
                ensure
                        Result /= Void

end -- class REMOVE_PHASE
```

b) tracker-definition.txt and acceptance tests:

tracker-definition.txt:

```
system tracker
-- tracking system for a nuclear plant

type CID = STRING -- container identifiers
type PID = STRING -- phase identifiers

type MATERIAL = {glass, metal, plastic, liquid}

type CONTAINER = TUPLE[material: MATERIAL; radioactivity: VALUE]

-- create a new tracking system
new_tracker(
            max_phase_radiation: VALUE
                -- max radiation allowed for all containers in a phase
        ; max_container_radiation: VALUE
                -- max radation allowed in a container
        )

-- create a new phase
new_phase(
            pid: PID                    -- phase identifier
        ; phase_name: STRING
        ; capacity: INT             -- number of containers phase handels
```

```
        ; expected_materials: ARRAY[MATERIAL]    -- subset of materials
    )

-- remove a phase with phase identifier pid
remove_phase(pid: PID)

-- create a new container and move it to a phase
new_container (
            cid: CID    -- container identifier
        ; c: CONTAINER
        ; pid:PID
        )

-- remove a container with container identifier cid
remove_container (cid: CID)

-- move a container from source phase to target phase
move_container (
            cid: CID      -- container identifier to be moved
        ; pid1: PID    -- source phase
        ; pid2: PID    -- target phase
        )

--undo/redo up to new_tracker
undo

redo
```

at1.expected.txt:

```
   state 0 ok
   max_phase_radiation: 0.00, max_container_radiation: 0.00
   phases: pid->name:capacity,count,radiation
   containers: cid->pid->material,radioactivity
->new_tracker(50,10)
   state 1 ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
   containers: cid->pid->material,radioactivity
->new_phase("pid2","compacting",2,<<glass, metal, plastic>>)
   state 2 ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid2->compacting:2,0,0.00,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
->new_phase("pid1","unpacking",2,<<glass, metal, plastic, liquid>>)
   state 3 ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,0,0.00,{glass,metal,plastic,liquid}
```

```
      pid2->compacting:2,0,0.00,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
->new_container("cid4",[metal, 3],"pid1")
   state 4 ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,1,3.00,{glass,metal,plastic,liquid}
      pid2->compacting:2,0,0.00,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
      cid4->pid1->metal,3.00
->new_container("cid1",[glass, 5.5],"pid1")
   state 5 ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,2,8.50,{glass,metal,plastic,liquid}
      pid2->compacting:2,0,0.00,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
      cid1->pid1->glass,5.50
      cid4->pid1->metal,3.00
->new_container("cid2",[liquid, 0.5],"pid1")
   state 6 e11: this container will exceed phase capacity
->move_container("cid1","pid1","pid2")
   state 7 ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,1,3.00,{glass,metal,plastic,liquid}
      pid2->compacting:2,1,5.50,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
      cid1->pid2->glass,5.50
      cid4->pid1->metal,3.00
->move_container("cid4","pid1","pid2")
   state 8 ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,0,0.00,{glass,metal,plastic,liquid}
      pid2->compacting:2,2,8.50,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
      cid1->pid2->glass,5.50
      cid4->pid2->metal,3.00
```

at2.expected.txt:

```
   state 0 ok
   max_phase_radiation: 0.00, max_container_radiation: 0.00
   phases: pid->name:capacity,count,radiation
   containers: cid->pid->material,radioactivity
->new_tracker(39,10)
   state 1 ok
   max_phase_radiation: 39.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
```

```
   containers: cid->pid->material,radioactivity
->new_tracker(-1,10)
   state 2 e2: max phase radiation must be non-negative value
->new_tracker(50,-1)
   state 3 e3: max container radiation must be non-negative value
->new_tracker(1,1.1)
   state 4 e4: max container must not be more than max phase radiation
->new_tracker(0,1)
   state 5 e4: max container must not be more than max phase radiation
->new_phase("pid3","compacting",2,<<glass, metal, plastic>>)
   state 6 ok
   max_phase_radiation: 39.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid3->compacting:2,0,0.00,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
->new_phase("pid2","assay",3,<<glass, metal, plastic, liquid>>)
   state 7 ok
   max_phase_radiation: 39.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid2->assay:3,0,0.00,{glass,metal,plastic,liquid}
      pid3->compacting:2,0,0.00,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
->new_phase("","assay",3,<<glass, metal, plastic, liquid>>)
   state 8 e5: identifiers/names must start with A-Z, a-z or 0..9
->new_phase("pid2","assay",3,<<glass, metal, plastic, liquid>>)
   state 9 e6: phase identifier already exists
->new_phase("pid9","",3,<<glass, metal, plastic, liquid>>)
   state 10 e5: identifiers/names must start with A-Z, a-z or 0..9
->new_phase("pid9","junk",0,<<glass, metal, plastic, liquid>>)
   state 11 e7: phase capacity must be a positive integer
->new_phase("pid9","junk",1,<<>>)
   state 12 e8: there must be at least one expected material for this phase
->new_phase("pid1","unpacking",4,<<glass, metal, plastic, liquid>>)
   state 13 ok
   max_phase_radiation: 39.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:4,0,0.00,{glass,metal,plastic,liquid}
      pid2->assay:3,0,0.00,{glass,metal,plastic,liquid}
      pid3->compacting:2,0,0.00,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
->new_phase("pid4","storage",7,<<glass, metal, plastic, liquid>>)
   state 14 ok
   max_phase_radiation: 39.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:4,0,0.00,{glass,metal,plastic,liquid}
      pid2->assay:3,0,0.00,{glass,metal,plastic,liquid}
      pid3->compacting:2,0,0.00,{glass,metal,plastic}
      pid4->storage:7,0,0.00,{glass,metal,plastic,liquid}
   containers: cid->pid->material,radioactivity
```

at3.expected.txt:

```
   state 0 ok
   max_phase_radiation: 0.00, max_container_radiation: 0.00
   phases: pid->name:capacity,count,radiation
   containers: cid->pid->material,radioactivity
->new_tracker(50,10)
   state 1 ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
   containers: cid->pid->material,radioactivity
->new_phase("pid2","compacting",2,<<glass, metal, plastic>>)
   state 2 ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid2->compacting:2,0,0.00,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
->new_phase("pid1","unpacking",2,<<glass, metal, plastic, liquid>>)
   state 3 ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,0,0.00,{glass,metal,plastic,liquid}
      pid2->compacting:2,0,0.00,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
->new_container("cid4",[metal, 3],"pid1")
   state 4 ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,1,3.00,{glass,metal,plastic,liquid}
      pid2->compacting:2,0,0.00,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
      cid4->pid1->metal,3.00
->new_container("cid1",[glass, 5.5],"pid1")
   state 5 ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,2,8.50,{glass,metal,plastic,liquid}
      pid2->compacting:2,0,0.00,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
      cid1->pid1->glass,5.50
      cid4->pid1->metal,3.00
->new_container("cid2",[liquid, 0.5],"pid1")
   state 6 e11: this container will exceed phase capacity
->move_container("cid1","pid1","pid2")
   state 7 ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,1,3.00,{glass,metal,plastic,liquid}
      pid2->compacting:2,1,5.50,{glass,metal,plastic}
```

```
   containers: cid->pid->material,radioactivity
      cid1->pid2->glass,5.50
      cid4->pid1->metal,3.00
->move_container("cid4","pid1","pid2")
   state 8 ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,0,0.00,{glass,metal,plastic,liquid}
      pid2->compacting:2,2,8.50,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
      cid1->pid2->glass,5.50
      cid4->pid2->metal,3.00
->new_container("cid4",[metal, 3],"pid1")
   state 9 e10: this container identifier already in tracker
->new_container("cid1",[glass, 5.5],"pid1")
   state 10 e10: this container identifier already in tracker
->new_container("cid2",[liquid, 0.5],"pid1")
   state 11 ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,1,0.50,{glass,metal,plastic,liquid}
      pid2->compacting:2,2,8.50,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
      cid1->pid2->glass,5.50
      cid2->pid1->liquid,0.50
      cid4->pid2->metal,3.00
->undo
   state 12 (to 10) e10: this container identifier already in tracker
->undo
   state 13 (to 9) e10: this container identifier already in tracker
->undo
   state 14 (to 8) ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,0,0.00,{glass,metal,plastic,liquid}
      pid2->compacting:2,2,8.50,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
      cid1->pid2->glass,5.50
      cid4->pid2->metal,3.00
->undo
   state 15 (to 7) ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,1,3.00,{glass,metal,plastic,liquid}
      pid2->compacting:2,1,5.50,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
      cid1->pid2->glass,5.50
      cid4->pid1->metal,3.00
->undo
```

```
   state 16 (to 6) e11: this container will exceed phase capacity
->undo
   state 17 (to 5) ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,2,8.50,{glass,metal,plastic,liquid}
      pid2->compacting:2,0,0.00,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
      cid1->pid1->glass,5.50
      cid4->pid1->metal,3.00
->undo
   state 18 (to 4) ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,1,3.00,{glass,metal,plastic,liquid}
      pid2->compacting:2,0,0.00,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
      cid4->pid1->metal,3.00
->undo
   state 19 (to 3) ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,0,0.00,{glass,metal,plastic,liquid}
      pid2->compacting:2,0,0.00,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
->undo
   state 20 (to 2) ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid2->compacting:2,0,0.00,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
->undo
   state 21 (to 1) ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
   containers: cid->pid->material,radioactivity
->undo
   state 22 e19: there is no more to undo
->undo
   state 23 e19: there is no more to undo
->redo
   state 24 (to 2) ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid2->compacting:2,0,0.00,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
->redo
   state 25 (to 3) ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
```

```
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,0,0.00,{glass,metal,plastic,liquid}
      pid2->compacting:2,0,0.00,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
->redo
   state 26 (to 4) ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,1,3.00,{glass,metal,plastic,liquid}
      pid2->compacting:2,0,0.00,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
      cid4->pid1->metal,3.00
->redo
   state 27 (to 5) ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,2,8.50,{glass,metal,plastic,liquid}
      pid2->compacting:2,0,0.00,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
      cid1->pid1->glass,5.50
      cid4->pid1->metal,3.00
->redo
   state 28 (to 6) e11: this container will exceed phase capacity
->redo
   state 29 (to 7) ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,1,3.00,{glass,metal,plastic,liquid}
      pid2->compacting:2,1,5.50,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
      cid1->pid2->glass,5.50
      cid4->pid1->metal,3.00
->redo
   state 30 (to 8) ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
      pid1->unpacking:2,0,0.00,{glass,metal,plastic,liquid}
      pid2->compacting:2,2,8.50,{glass,metal,plastic}
   containers: cid->pid->material,radioactivity
      cid1->pid2->glass,5.50
      cid4->pid2->metal,3.00
->redo
   state 31 (to 9) e10: this container identifier already in tracker
->redo
   state 32 (to 10) e10: this container identifier already in tracker
->redo
   state 33 (to 11) ok
   max_phase_radiation: 50.00, max_container_radiation: 10.00
   phases: pid->name:capacity,count,radiation
```

pid1->unpacking:2,1,0.50,{glass,metal,plastic,liquid}
pid2->compacting:2,2,8.50,{glass,metal,plastic}
containers: cid->pid->material,radioactivity
cid1->pid2->glass,5.50
cid2->pid1->liquid,0.50
cid4->pid2->metal,3.00
->redo
state 34 e20: there is no more to redo
->redo
state 35 e20: there is no more to redo