# EECS3311-W2017 — Project Report

Submitted electronically by:

| Team members | Name | Prism Login | Signature |
|---|---|---|---|
| Member 1: | Mikhail Gindin | mikegin | *Mikhail Gindin* |
| Member 2: | | | |
| *Submitted under Prism account: | | mikegin | |

* Submit under **one** Prism account only

**Contents**


1. Requirements for Invoicing System....6
2. BON class diagram overview (archecturite of the design)....7
3. Table of modules — responsibilities and information hiding....8
4. Expanded description of design decisions....9
5. Significant Contracts (Correctness)....10
6. Summary of Testing Procedures....11
7. Appendix (Contract view of all classes)....12

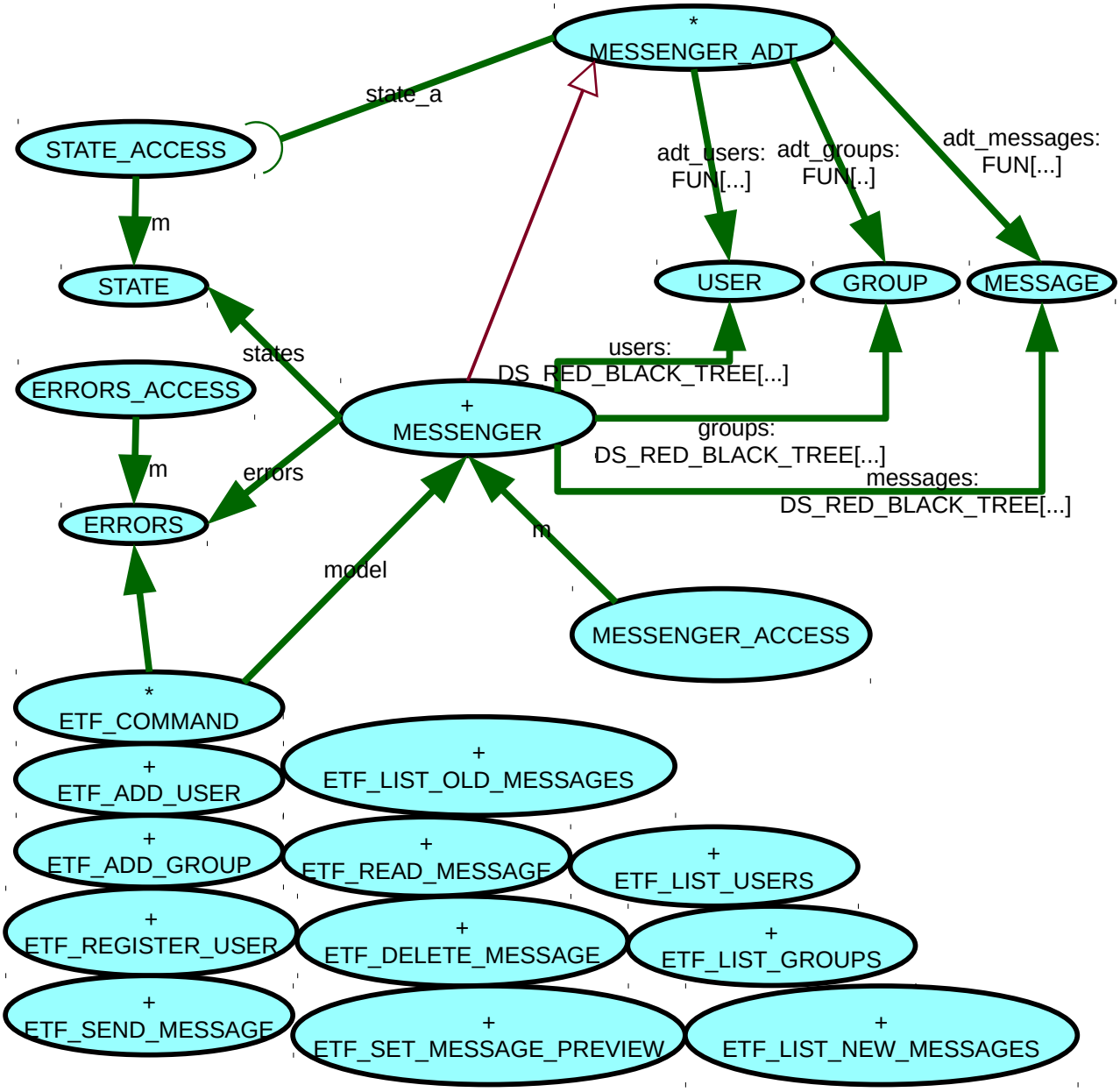
## 1. Requirements for Project “Messenger”

Management has decided to develop a messaging system for use in hospitals. Physicians, nurses, and administrators shall be able to securely use it for communication. Users can be subscribed to groups and can send a message to the group. It is essential for privacy concerns that only those users registered in a group may read messages directed at the group. When an operation can be performed, the system responds with “ok”. However, when some operation cannot be performed, a meaningful error message is displayed. A console based application or user input suffices.

See *messenger.definition.txt*, *errors.txt*, *at1.expected.txt*, and the oracle provided to us for implementation details and expected output.

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

### a)



## b)

The design revolved around a MESSENGER class. This was the main class that would house all the users, groups, and messages in the system and implement the majority of the business logic and output. The MESSENGER_ADT was an abstract version of MESSENGER that MESSENGER inherited from. It housed all of MESSENGER's contracts, and used the math models FUN and REL classes to simplify contracts like function override. MESSENGER is accessible only through MESSENGER_ACCESS, using the Singleton design pattern, since only one instance of MESSENGER is needed.

Users are represented as USER objects and likewise, groups are represented as GROUP objects. They house information like their ids and names. A USER object also stores information about what groups the user is associated with and an inbox which houses what messages the user has received/sent with and their respective states. A GROUP object also stores information about its members and what messages have been sent to the group. The attributes of USER and GROUP are hidden and are accessible only to MESSENGER and MESSENGER_ADT.

There is also a MESSAGE class, which represents a sent message. It houses information like the message id, the message preview size, and the message content.

There is an ERRORS class which houses all error messages and warnings. It is accessible through ERRORS_ACCESS, using the Singleton design pattern, since multiple ERRORS objects are unnecessary. Likewise there is a STATE class which houses all the message states. Like ERRORS, it is accessible through STATE_ACCESS using the Singleton design pattern.

Finally there are the ETF_COMMAND classes which represent user commands. They all call MESSENGER for basic error checking information and call MESSENGER for command implementation.

## 3. Table of modules — responsibilities and information hiding

| | | |
|---|---|---|
| 1 | MESSENGER_ADT | **Responsibility:** Class that houses the main business logic and output of the messenger application. |
| | Abstract | **Secret:** Uses math models to model addition of users, groups, messages, and registrations. |

| | | |
|---|---|---|
| 1.1 | MESSENGER | **Responsibility:** See MESSENGER_ADT. |
| | Concrete | **Secret:** Uses Red Black Trees to store users, groups, and messages for efficient insert and delete operations, sorted by ids. Stores a second user tree and group tree for name sorting. Implements the execution of the abstract math models alongside the concrete implementation so both coincide. |

| | | |
|---|---|---|
| 1.2 | MESSENGER_ACCESSOR | **Responsibility:** Accessor to MESSENGER |
| | Concrete | **Secret:** None |

| | | |
|---|---|---|
| 2 | USER | **Responsibility:** Stores information about a user. |
| | Concrete | **Secret:** Uses a sorted list for storing group id' and a Red Black tree for storing message states, sorted by message ids. |

| | | |
|---|---|---|
| 3 | Group | **Responsibility:** Stores information about a group. |
| | Concrete | **Secret:** Uses a sorted list for storing group ids and a sorted list for storing message ids. |

| | | |
|---|---|---|
| 4 | MESSAGE | **Responsibility:** Stores information about a message. |
| | Concrete | **Secret:** None |

| | | |
|---|---|---|
| 5 | ERRORS | **Responsibility:** Stores list of errors and warnings. |
| | Concrete | **Secret:** None |

| | | |
|---|---|---|
| 5.1 | ERRORS_ACCESS | **Responsibility:** Accessor to the ERRORS class. |
| | Concrete | **Secret:** None. |

| | | |
|---|---|---|
| 6 | STATE | **Responsibility:** Stores possible message states. |
| | Concrete | **Secret:** None. |

| | | |
|---|---|---|
| 6.1 | STATE_ACCESS | **Responsibility:** Accessor to the STATE class. |
| | Concrete | **Secret:** None. |

## 4. Expanded description of design decisions

The MESSENGER class is the main class behind the business logic of the program. It houses all the users, groups, and messages of the system. It also contains implementation for client commands/queries and deals with output formatting.

To house the users, groups and messages, Red Black trees were picked for efficiency and sorting. Insert operations were O(logn) and traversal was O(n). The trees were sorted by id, but there were also an additional user and group tree that would sort by USER and GROUP objects respectively. This was for the list_user and list_group commands that required sorting by name.

Add_user and add_group would create USER and GROUP objects and add them to the user and group trees respectively. Register_user would add the user to the GROUP object's users list, and would add the group to the USERS object's groups list. This would represent a registration. Send_message would do two things. First it stores the message in the messages tree. Second, it sends the message to the inbox of each user in the GROUP object's users list with the correct state. Read_message changes a user's message state. Delete_message removes the message from the user's inbox.

List_users, list_groups, list_new_messages, list_old_messages all modify output variables by using the user, group, and message trees.

MESSENGER also had a variety of queries used for contracts, and error checking by the ETF_COMMAND classes. For example, registration_exists would check if a user was registered in a certain group and message_unavailable would check if a user registered in a group does not have a message that was sent to it (i.e if it was sent prior to the user joining the group).

MESSENGER contains some helper functions, to help some commands and also for dealing with the output. The output is split into chunks, each dealing with a certain type of output, and properly concatenated in the out function. MESSENGER also contains some setter methods for setting the required errors and warnings, called by the ETF_COMMAND classes.

## 5. Significant Contracts (Correctness)

.

MODEL_ADT was the class that housed the contracts for MESSENGER. It would use the math models FUN and REL classes to model the tree's in MESSENGER. MESSENGER_ADT would also use queries implemented in MESSENGER for various command preconditions.

Add_user and add_group commands used the adt_users and adt_groups FUN objects, where in the post condition the function override operator was used to ensure that users and groups were properly added. In register_user, the adt_registrations REL object was used to ensure that the new user – group mapping exists. Send_message ensured that the message was added to adt_messages FUN object, that the message was added to the GROUP object's messages list, and that the message was correctly sent to the group's users. Read_message ensured that the user's message in their inbox was changed correctly, and delete_message ensured that the user no longer had the message in their inbox. Set_message_preview checked that the message preview settings were properly attributed in all the messages.

## 6. Summary of Testing Procedures

### a)

| Test file | Description | Passed |
|---|---|---|
| at1.txt | Testing adding a user and a group, registrations, and message sending. | Yes |
| at2.txt | Testing message reading. | Yes |
| at3.txt | Testing message deleting | Yes |
| at4.txt | Testing setting the message preview. | Yes |
| at5.txt | Testing list_new_messages | Yes |
| at6.txt | Testing list_old_messages | Yes |
| at7.txt | Testing list_users | Yes |
| at8.txt | Testing list_groups | Yes |
| at9.txt | Random commands. | Yes |
| at1.txt (instructor) | Various operations. | Yes |

**b)**

**Test Run:04/01/2017 6:56:44.313 PM**

# ROOT

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** | STUDENT_TESTS | |
| **PASSED** | NONE | t1: Test add_user |
| **PASSED** | NONE | t2: Test add_group |
| **PASSED** | NONE | t3: Test register_user |
| **PASSED** | NONE | t4: Test send_message |
| **PASSED** | NONE | t5: Test read_message |
| **PASSED** | NONE | t6: Test delete_message |
| **PASSED** | NONE | t7: Test set_message_preview<br><br>sender: 2, group: 4, content: "My message is more t..." |
| **PASSED** | NONE | t8: Test has_old_message and has_new_message |

# 7. Appendix (Contract view of all classes)

```
note
    description: "Class housing errors and warnings"
    author: "Mikhail Gindin"
    date: "$Date$"
    revision: "$Revision$"

    class interface
    ERRORS

    create
    make

    feature -- list of errors

    Ok: STRING_8 = "OK"

    Error: STRING_8 = "ERROR "

    Id_must_be_positive: STRING_8 = "ID must be a positive integer."

    Id_in_use: STRING_8 = "ID already in use."

    User_name_starts_with_letter: STRING_8 = "User name must start with a letter."

    Group_name_starts_with_letter: STRING_8 = "Group name must start with a letter."

    User_not_exists: STRING_8 = "User with this ID does not exist."

    Group_not_exists: STRING_8 = "Group with this ID does not exist."

    Registration_already_exists: STRING_8 = "This registration already exists."

    Message_empty: STRING_8 = "A message may not be an empty string."

    Not_authorized_group: STRING_8 = "User not authorized to send messages to the
    specified group."

    Message_not_exists: STRING_8 = "Message with this ID does not exist."

    User_not_authorized: STRING_8 = "User not authorized to access this message."

    Message_id_unavailable: STRING_8 = "Message with this ID unavailable."

    Already_read: STRING_8 = "Message has already been read. See `list_old_messages'."
```

```
Old_message_not_found: STRING_8 = "Message with this ID not found in old/read
messages."

Incorrect_message_length: STRING_8 = "Message length must be greater than zero."

feature -- list of warnings

W_no_new_messages: STRING_8 = "There are no new messages for this user."

W_no_old_messages: STRING_8 = "There are no old messages for this user."

W_no_groups_registered: STRING_8 = "There are no groups registered in the system
yet."

W_no_users_registered: STRING_8 = "There are no users registered in the system
yet."

end -- class ERRORS
```

```
note
description: "ERRORS accessor."
author: "Mikhail Gindin"
date: "$Date$"
revision: "$Revision$"

expanded class interface
ERRORS_ACCESS

create
default_create

feature

M: ERRORS

invariant
        M = M

end -- class ERRORS_ACCESS
```

```
note
    description: "Class that houses possible message states."
    author: ""
    date: "$Date$"
    revision: "$Revision$"

    class interface
    STATE

    create {STATE_ACCESS}
    make

    feature -- Attributes

    read: STRING_8

    unread: STRING_8

    unavailable: STRING_8

    end -- class STATE




    note
    description: "STATE accessor."
    author: "Mikhail Gindin"
    date: "$Date$"
    revision: "$Revision$"

    expanded class interface
    STATE_ACCESS

    create
    default_create

    feature

    M: STATE

    invariant
            M = M

    end -- class STATE_ACCESS
```

```
note
    description: "Class that represents a user."
    author: "Mikhail Gindin"
    date: "$Date$"
    revision: "$Revision$"

    class interface
    USER

    create
    make

    end -- class USER




    note
    description: "Class that represents a group."
    author: "Mikhail Gindin"
    date: "$Date$"
    revision: "$Revision$"

    class interface
    GROUP

    create
    make

    end -- class GROUP




    note
    description: "Class that represents a message."
    author: "Mikhail Gindin"
    date: "$Date$"
    revision: "$Revision$"

    class interface
    MESSAGE

    create
    make

    feature -- Queries

    get_id: INTEGER_64

    get_txt: STRING_8

    get_sender: INTEGER_64

    get_group: INTEGER_64

    get_msg_prev: INTEGER_64
```

```
    feature -- Commands

    set_message_prev (n: INTEGER_64)

    feature -- output

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

    end -- class MESSAGE
```

```
note
    description: "Abstract Messenger that handles contracting."
    author: "Mikhail Gindin"
    date: "$Date$"
    revision: "$Revision$"

    deferred class interface
    MESSENGER_ADT

    feature -- Attributes

    adt_users: FUN [INTEGER_64, USER]
                -- abstract users storage

    adt_groups: FUN [INTEGER_64, GROUP]
                -- abstract groups storage

    adt_messages: FUN [INTEGER_64, MESSAGE]
                -- abstract messages storage

    adt_registrations: REL [INTEGER_64, INTEGER_64]
                -- abstract registrations storage

    mid_counter: INTEGER_64
                -- message id counter

    state_a: STATE_ACCESS

    feature -- Commands

    add_user (uid: INTEGER_64; user_name: STRING_8)
                --adds a user to the system
            require
                        id_positive (uid)
                        not user_exists (uid)
                        name_starts_with_letter (user_name)
            ensure
                        adt_users ~ old adt_users @<+ [uid, create {USER}.make (uid,
```

```
                        user_name)]

add_group (gid: INTEGER_64; group_name: STRING_8)
            --adds a group to the system
        require
                    id_positive (gid)
                    not group_exists (gid)
                    name_starts_with_letter (group_name)
        ensure
                    adt_groups ~ old adt_groups @<+ [gid, create {GROUP}.make
                    (gid, group_name)]

register_user (uid: INTEGER_64; gid: INTEGER_64)
            --registers a user in a group
        require
                    id_positive (uid) and id_positive (gid)
                    user_exists (uid)
                    group_exists (gid)
                    not registration_exists (uid, gid)
        ensure
                    adt_registrations ~ old adt_registrations + create {PAIR
                    [INTEGER_64, INTEGER_64]}.make_from_tuple ([uid,
            gid])

send_message (uid: INTEGER_64; gid: INTEGER_64; txt: STRING_8)
            --sends a message from the user to the users in the group
        require
                    id_positive (uid) and id_positive (gid)
                    user_exists (uid)
                    group_exists (gid)
                    not message_empty (txt)
                    registration_exists (uid, gid)
        ensure
                    adt_messages ~ old adt_messages + create {PAIR [INTEGER_64,
                            MESSAGE]}.make_from_tuple ([mid_counter - 1, create
                    {MESSAGE}.make (mid_counter - 1, uid, gid, txt)])
            group_has_message: adt_groups [gid].has_message (mid_counter - 1)
            correctly_sent: across
                            adt_groups [gid].get_users as cr
                    all
                            adt_users [cr.item].get_id = uid implies adt_users
                            [cr.item].get_from_inbox (mid_counter - 1) ~
                            state_a.M.read and adt_users [cr.item].get_id
                            /= uid implies adt_users
                    [cr.item].get_from_inbox (mid_counter - 1) ~
                    state_a.M.unread
                    end

read_message (uid: INTEGER_64; mid: INTEGER_64)
            --reads the user's message
        require
                    id_positive (uid) and id_positive (mid)
                    user_exists (uid)
                    message_exists (mid)
                    authorized_message_access (uid, mid)
                    not message_unavailable (uid, mid)
                    not message_already_read (uid, mid)
        ensure
                    adt_users [uid].get_from_inbox (mid) ~ state_a.M.read
```

```
delete_message (uid: INTEGER_64; mid: INTEGER_64)
                --deletes the user's message
        require
                        id_positive (uid) and id_positive (mid)
                        user_exists (uid)
                        message_exists (mid)
                        message_id_found_in_old_messages (uid, mid)
        ensure
                        not adt_users [uid].has_in_inbox (mid)

set_message_preview (n: INTEGER_64)
                --sets the message preview character amount across all messages
        require
                        correct_message_length (n)
        ensure
                        across
                                adt_messages as cr
                        all
                                adt_messages.item (cr.item.first).get_msg_prev = n
                        end

feature -- output Commands

list_new_messages (uid: INTEGER_64)
                --formats the output to list the new messages of the user
        require
                        id_positive (uid)
                        user_exists (uid)
                        has_new_messages (uid)

list_old_messages (uid: INTEGER_64)
                --formats the output to list the old messages of the user
        require
                        id_positive (uid)
                        user_exists (uid)
                        has_old_messages (uid)

list_users
                --formats the output to list all users

list_groups
                --formats the output to list all groups

feature -- Queries

id_positive (id: INTEGER_64): BOOLEAN
                --checks if the id is positive

name_starts_with_letter (name: STRING_8): BOOLEAN
                --checks if the name starts with a letter

user_exists (id: INTEGER_64): BOOLEAN
                --checks if the user exists

group_exists (id: INTEGER_64): BOOLEAN
                --checks if the group exists

message_exists (id: INTEGER_64): BOOLEAN
```

```
                --checks if the message exists

registration_exists (uid: INTEGER_64; gid: INTEGER_64): BOOLEAN
                --checks if the user is registered in the group
        require
                        user_exists (uid) and group_exists (gid)

message_empty (txt: STRING_8): BOOLEAN
                --checks if a message is empty

authorized_message_access (uid: INTEGER_64; mid: INTEGER_64): BOOLEAN
                --checks if the user is authorized to access the message
        require
                        user_exists (uid)
                        message_exists (mid)

message_unavailable (uid: INTEGER_64; mid: INTEGER_64): BOOLEAN
                --checks if the message is unavailable to the user
        require
                        user_exists (uid)
                        message_exists (mid)

message_already_read (uid: INTEGER_64; mid: INTEGER_64): BOOLEAN
                --checks if the user's message is read
        require
                        message_exists (mid)
                        authorized_message_access (uid, mid)

message_id_found_in_old_messages (uid: INTEGER_64; mid: INTEGER_64): BOOLEAN
                --checks if the mid is found in the users inbox
        require
                        user_exists (uid)
                        message_exists (mid)

correct_message_length (n: INTEGER_64): BOOLEAN
                --checks if the message length is correct

has_old_messages (uid: INTEGER_64): BOOLEAN
                --checks if the user has old messages

has_new_messages (uid: INTEGER_64): BOOLEAN
                --checks if the user has new messages

users_exist: BOOLEAN
                --checks if a user exists

groups_exist: BOOLEAN
                --checks if a group exists

end -- class MESSENGER_ADT
```

```
note
	description: "Class responsible for the main business logic of the Messenger app."
	author: "Mikhail Gindin"
	date: "$Date$"
	revision: "$Revision$"

class interface
	MESSENGER

create {MESSENGER_ACCESS}
	make

feature -- Commands

	reset
			-- Reset model state.

	add_user (uid: INTEGER_64; user_name: STRING_8)
			--adds a user to the system

	add_group (gid: INTEGER_64; group_name: STRING_8)
			--adds a group to the system

	register_user (uid: INTEGER_64; gid: INTEGER_64)
			--registers a user in a group

	send_message (uid: INTEGER_64; gid: INTEGER_64; txt: STRING_8)
			--sends a message from the user to the users in the group

	read_message (uid: INTEGER_64; mid: INTEGER_64)
			--reads the user's message

	delete_message (uid: INTEGER_64; mid: INTEGER_64)
			--deletes the user's message

	set_message_preview (n: INTEGER_64)
			--sets the message preview character amount across all messages

feature -- Queries

	id_positive (id: INTEGER_64): BOOLEAN
			--checks if the id is positive

	name_starts_with_letter (name: STRING_8): BOOLEAN
			--checks if the name starts with a letter

	user_exists (id: INTEGER_64): BOOLEAN
			--checks if the user exists

	group_exists (id: INTEGER_64): BOOLEAN
			--checks if the group exists
```

```
        message_exists (id: INTEGER_64): BOOLEAN
                        --checks if the message exists

        registration_exists (uid: INTEGER_64; gid: INTEGER_64): BOOLEAN
                        --checks if the user is registered in the group

        message_empty (txt: STRING_8): BOOLEAN
                        --checks if a message is empty

        authorized_message_access (uid: INTEGER_64; mid: INTEGER_64): BOOLEAN
                        --checks if the user is authorized to access the message

        message_unavailable (uid: INTEGER_64; mid: INTEGER_64): BOOLEAN
                        --checks if the message is unavailable to the user

        message_already_read (uid: INTEGER_64; mid: INTEGER_64): BOOLEAN
                        --checks if the user's message is read

        message_id_found_in_old_messages (uid: INTEGER_64; mid: INTEGER_64): BOOLEAN
                        --checks if the mid is found in the users inbox

        correct_message_length (n: INTEGER_64): BOOLEAN
                        --checks if the message length is correct

        has_old_messages (uid: INTEGER_64): BOOLEAN
                        --checks if the user has old messages

        has_new_messages (uid: INTEGER_64): BOOLEAN
                        --checks if the user has new messages

        users_exist: BOOLEAN
                        --checks if a user exists

        groups_exist: BOOLEAN
                        --checks if a group exists

feature -- output Commands

        list_new_messages (uid: INTEGER_64)
                        --formats the output to list the new messages of the user

        list_old_messages (uid: INTEGER_64)
                        --formats the output to list the old messages of the user

        list_users
                        --formats the output to list all users

        list_groups
                        --formats the output to list all groups
```

```
        set_query_warning (s: STRING_8)

        set_query_list_warning (s: STRING_8)

        set_error_message (s: STRING_8)

feature -- output Queries

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

invariant
        user_trees_coincide:

                across
                        users as cr
                all
                        users_name.has (cr.item)
                end
                across
                        users_name as cr
                all
                        users.has (cr.item.get_id)
                end
        group_trees_coincide:

        across
                        groups as cr
                all
                        groups_name.has (cr.item)
                end
                across
                        groups_name as cr
                all
                        groups.has (cr.item.get_id)
                end

        end -- class MESSENGER
```

```
note
    description: "Singleton access to the default business model."
    author: "Jackie Wang"
    date: "$Date$"
    revision: "$Revision$"

    expanded class interface
    MESSENGER_ACCESS

    create
    default_create

    feature

    M: MESSENGER

    invariant
            M = M

    end -- class MESSENGER_ACCESS
```