Study Actions

This document describes the currently available study actions provided by CASE.

Study actions are performed by using the following collection of `struct` objects as parameters:

- `action` : an object of `types.Expression`. Specifies the actions that should be performed.
- `oldState` : an object of `types.ParticipantState`. Characterizes the current state of the participant that will potentially be updated by evaluating the action expression.
- `event` : an object of `types.StudyEvent`. Specifies the event that was triggered by the participant or by the program (e.g. `"TIMER"`) and collects the survey responses,
- `dbService` : an object of `types.StudyDBService`. References the database abstraction layer to get access to previous responses of the participant (for example to check them for conditions specified by the researcher).

The functions executing actions are listed in the following. The header denotes the string keyword leading to the decision which kind of action will be performed. The block code indicates the header of the function that will be executed in case of the keyword specified.

1. IF

```
ifAction(action, oldState, event, dbService)
```

This function is used for control flow implementing the typical if-else structure. Conditions are checked in order to decide which actions will be performed.

Required Parameter in this function:

> `action.Data[0]` : condition, evaluated to a boolean value
> `action.Data[1]` : perform this action if condition is true
> `action.Data[2]` : (optional) perform this action otherwise

Note: If the execution of action fails, the old state of the participant is returned unmodified. The length of `action.Data` must be at least 2.

Return: `(types.ParticipantState, error)`

2. DO

```
doAction(action, oldState, event, dbService)
```

performs a list of actions by iterating through the `action.Data` argument. This function can be used to group actions together as a defined list of arguments.

Required Parameter in this function:

> `action.Data[0:]` : list of actions that will be performed successively

Return: `(types.ParticipantState, error)`

3. IFTHEN

```
ifThenAction(action, oldState, event, dbService)
```

conditionally performs a list of actions. The function checks the first entry of the `action.Data` argument as condition deciding if actions will be performed. In case of `true` it iterates through the following entries of the `action.Data` argument.

Required Parameter in this function:

> `action.Data[0]` : condition, evaluated to a boolean value
> `action.Data[1:]` : perform this sequence of actions if condition is true

Note: The length of `action.Data` must be at least 1.

Return: `(types.ParticipantState, error)`

4. UPDATE_STUDY_STATUS

```
updateStudyStatusAction(action, oldState, event)
```

updates the status of the participant (e.g. from active to inactive). possible status values: "active", "inactive", "paused", "finished")

Required Parameter in this method:

> `action.Data[0]` : the new status of the participant convertible to `string`. Possible values are: `"active"`, `"inactive"`, `"paused"`, `"finished"`.

Note: The length of `action.Data` must be 1.

Return: `(types.ParticipantState, error)`

5. UPDATE_FLAG

```
updateFlagAction(action, oldState, event)
```

updates one flag of the participant state. The flag attribute of the `ParticipantState` object is a map with string keys addressing corresponding string values.

Required Parameter in this method:

> `action.Data[0]` : the string key of the flag to be updated
> `action.Data[1]` : the string value of the flag to be updated

Note: The length of `action.Data` must be 2.

Return: `(types.ParticipantState, error)`

6. REMOVE_FLAG

```
removeFlagAction(action, oldState, event)
```

deletes the flag with the specified key of the participant state. The flag attribute of the `ParticipantState` object is a map with string keys addressing corresponding string values.

Required Parameter in this method:

> `action.Data[0]` : the string key of the flag to be removed

Note: The length of `action.Data` must be 1.

Return: `(types.ParticipantState, error)`

7. ADD_NEW_SURVEY

```
addNewSurveyAction(action, oldState, event)
```

appends a new survey to the assigned surveys of the participant state (expressed by the attribute `AssignedSurveys` of `ParticipantState`).

Required Parameter in this method:

> `action.Data[0]` : the string key of the survey to be assigned to the participant
> `action.Data[1]` : a float value indicating the timestamp from which the assigned survey is visible
> `action.Data[2]` : a float value indicating the time until the assigned survey is visible and should be submitted
> `action.Data[3]` : a string value indicating the mode of displaying the assigned survey. Possible values are `"prio"`, `"normal"`, `"quick"` or `"update"`.

Note: The length of `action.Data` must be 4.

Return: `(types.ParticipantState, error)`

8. REMOVE_ALL_SURVEYS

```
removeAllSurveys(action, oldState, event)
```

clears the list of assigned surveys of participant state.

Note: Arguments of `action.Data` are permitted for this function.

Return: `(types.ParticipantState, error)`

9. REMOVE_SURVEY_BY_KEY

```
removeSurveyByKey(action, oldState, event)
```

removes the first or last occurence of a survey with specific key in the list of assigned surveys of the participant state.

Required Parameter in this method:

> `action.Data[0]` : the string key of the survey to be removed at first or last occurence.
> `action.Data[1]` : a string value indicating if the first or last occurence of an asssigned survey should be removed. Expected values are `"first"` or `"last"`.

Note: The length of `action.Data` must be 2.

Return: `(types.ParticipantState, error)`

10. REMOVE_SURVEYS_BY_KEY

```
removeSurveysByKey(action, oldState, event)
```

removes all surveys with the specified key in the list of assigned surveys of the participant state.

Required Parameter in this method:

> `action.Data[0]` : the string key of the surveys to be removed.

Note: The length of `action.Data` must be 1.

Return: `(types.ParticipantState, error)`

11. ADD_REPORT

```
addReport(action, oldState, event)
```

finds and appends a response to a survey item (expressed by the `SurveyItemResponse` object) to the reports array of the participant state.

Required Parameter in this method:

> `action.Data[0]` : the string key of the survey item response to be added to reports array.

Note: The length of `action.Data` must be 1.

Return: `(types.ParticipantState, error)`

12. REMOVE_ALL_REPORTS

```
removeAllReports(action, oldState, event)
```

clears the reports array of participant state.

Note: Arguments of `action.Data` are permitted for this function.

Return: `(types.ParticipantState, error)`

13. REMOVE_REPORT_BY_KEY

```
removeReportByKey(action, oldState, event)
```

removes the first or last occurence of a survey item response with specific key in the list of reports of the participant state.

Required Parameter in this method:

> `action.Data[0]` : the string key of the survey item response to be removed at first or last occurence in reports.
> `action.Data[1]` : a string value indicating if the first or last occurence of an survey item response should be removed. Expected values are `"first"` or `"last"`.

Note: The length of `action.Data` must be 2.

Return: `(types.ParticipantState, error)`

14. REMOVE_REPORTS_BY_KEY

```
removeReportsByKey(action, oldState, event)
```

removes all survey item responses with the specified key in the list of reports of the participant state.

Required Parameter in this method:

> `action.Data[0]` : the string key of the survey item reponses to be removed.

Note: The length of `action.Data` must be 1.

Return: `(types.ParticipantState, error)`