MedDream DICOM Viewer Communication API

Add component to your project

Import and create new Viewer Communication component in your project:

```
const viewerCommunication = new ViewerCommunication(targetURL, integration);
```

Parameters:

- `targetURL` - MedDream Viewer URL;
- `integration` (Optional) - Integration type: `study` or `token`. Default value: `study`.

Window reference functions

Get available Viewer window reference

```
const windowReference = viewerCommunication.getWindowReference();
```

Focus available window

```
viewerCommunication.focusWindow();
```

Post action message to MedDream Viewer

```
viewerCommunication.postActionMessage(actionType, actionData);
```

Parameters:

- `actionType` - Action message command type;
- `actionData` - Data needed for action message.

For more details about available action messages check: `MedDream communication documentation`

Functions to open MedDream Viewer

Open studies in MedDream

```
viewerCommunication.openInMedDream(studies/token);
```

Parameters:

- `studies` (For `study` integration) - Study uid's list separated with `,`;
- `token` (For `token` integration) - Token with study information.

Add studies to MedDream

```
viewerCommunication.addToMedDream(studies/token);
```

Parameters:

- `studies` (For `study` integration) - Study uid's list separated with `,`;
- `token` (For `token` integration) - Token with study information.

Replace studies in MedDream

```
viewerCommunication.replaceInMedDream(studies/token);
```

Parameters:

- `studies` (For `study` integration) - Study uid's list separated with `,`;
- `token` (For `token` integration) - Token with study information.

Communication functions

Functions only for Study integration

Open study

```
viewerCommunication.openStudy(study);
```

Parameter:

- `study` - Study uid.

Open studies

```
viewerCommunication.openStudies(studies);
```

Parameter:

- `studies` - Array of study uid's.

Replace studies

```
viewerCommunication.replaceStudies(studies);
```

Parameter:

- `studies` - Array of study uid's.

Preload studies

```
viewerCommunication.preloadStudies(studies);
```

Parameter:

- `studies` - Array of study uid's.

Cache studies

```
viewerCommunication.cacheStudies(studies);
```

Parameter:

- `studies` - Array of study objects. Each study object has ***studyUid*** and ***storageId*** parameters.

Array example:

```
const studies = [
    {
        studyUid: 'study-uid-1',
        storageId: 'storage-id'
    },
    {
        studyUid: 'study-uid-2',
        storageId: 'storage-id'
    }
];
```

Close studies

```
viewerCommunication.closeStudies(studies);
```

Parameter:

- `studies` - Array of study objects. Each study object has ***studyUid*** and ***storageId*** parameters.

Array example:

```
const studies = [
    {
        studyUid: 'study-uid-1',
        storageId: 'storage-id'
    }
];
```

Functions only for Token integration

Open studies

```
viewerCommunication.openStudies(token);
```

Parameter:

- `token` - Generated token with studies information.

Replace studies

```
viewerCommunication.replaceStudies(token);
```

Parameter:

- `token` - Generated token with studies information.

Preload studies

```
viewerCommunication.preloadStudies(token);
```

Parameter:

- `token` - Generated token with studies information.

Cache studies

```
viewerCommunication.cacheStudies(token);
```

Parameter:

- `token` - Generated token with studies information.

Close studies

```
viewerCommunication.closeStudies(token);
```

Parameter:

- `token` - Generated token with studies information.

Common functions

Cache all studies

```
viewerCommunication.cacheAllStudies();
```

Close all studies

```
viewerCommunication.closeAllStudies();
```

Set layout

```
viewerCommunication.setLayout(columns, rows);
```

Parameters:

- `columns` - Number of columns;
- `rows` - Number of rows.

Open instance

```
viewerCommunication.openInstance(instanceUid, viewportColumn, viewportRow, viewportActions);
```

Parameters:

- `instanceUid` - Unique instance uid which has to be opened to viewport;
- `viewportColumn` - Column number of desired viewport;
- `viewportRow` - Row number of desired viewport;
- `viewportActions` - Object of actions which have to be performed on viewport after instance is loaded.

Available viewport actions:

- `windowing` - Windowing level. Available options: ***"DEFAULT"***, ***"AUTO"***, ***"CUSTOM"***. If ***"CUSTOM"*** windowing is selected, ***customWindowing*** parameter has to be defined in ***viewportActions*** object;
- `customWindowing` - Custom windowing level. This parameter allows to set custom windowing ***width*** and ***center*** levels. ***customWindowing*** has to be defined only when ***"CUSTOM"*** windowing is selected;
- `rotation` - Instance rotation by defined number of degrees;
- `verticalFlip` - Vertical instance flip. Available options: ***true***/***false***;
- `horizontalFlip` - Horizontal instance flip. Available options: ***true***/***false***;
- `scale` - Instance scaling option. Available options: ***"ORIGINAL"***, ***"FIT_TO_SCREEN"***, ***"CUSTOM"***. If ***"CUSTOM"*** scale is selected, ***customScale*** parameter has to be defined in ***viewportActions*** object.
- `customScale` - Custom scale number.
- `alignment` - Instance alignment in viewport. Available options: ***"RIGHT"***, ***"LEFT"***, ***"CENTER"***.

Viewport actions object example:

```
const viewportActions = {
    windowing: 'CUSTOM',                    //DEFAULT, AUTO, CUSTOM
    customWindowing: {width: 2, center: 2}, //Use if custom windowing
    rotation: 45,
    verticalFlip: true,
    horizontalFlip: true,
    scale: 'CUSTOM',                        //ORIGINAL, FIT_TO_SCREEN, CUSTOM
    customScale: 0.5,                       //Use if custom scale
    alignment: 'CENTER'                     //RIGHT, LEFT, CENTER
};
```

Export instance

```
viewerCommunication.exportInstance(viewportColumn, viewportRow);
```

Parameters:

- `viewportColumn` (Optional) - Column number of desired viewport;
- `viewportRow` (Optional) - Row number of desired viewport;

Currently active viewport instance is exported, if `viewportColumn` and `viewportRow` are not provided.

Get opened studies

```
const callback = (studies) => {console.log(studies)};
viewerCommunication.onGetOpenedStudies(callback);
viewerCommunication.getOpenedStudies();
```

Usage:

- Register ***onGetOpenedStudies callback*** function;
- Call ***getOpenedStudies*** function to request opened studies callback;
- Once message is processed, ***callback*** function will be triggered with opened studies array.