MedDream DICOM Viewer communication documentation

Product version: **8.2.0**

Document version: **1.10**

Date: **2023-03-07**

Contents

1. Introduction

MedDream communication service is based on JavaScript function Window.postMessage() and the capabilities that this function provides. To communicate with MedDream Viewer tab you will need to receive Viewer tab window reference and by using ***postMessage*** function provide needed commands to MedDream Viewer tab.

2. Set up

2.1. Configuration

Update his MedDream authentication configuration in ***application.properties*** file with code lines bellow:

```
authentication.his.valid-his-params=study //Use if study integration
authentication.his.useSameSession=true
```

In the same ***application.properties*** file add white list of domains from which communication with MedDream Viewer tab will be allowed:

```
security.postMessageWhitelist=DomainAddress1,DomainAddress2,...
```

2.2. Open MedDream Viewer

From your web solution open new MedDream Viewer tab by using:

- With study integration

```
const windowReference = window.open("TargetDomainAddress?study=studyUid",
"_blank");
```

- With token integration

```
const windowReference = window.open("TargetDomainAddress?token=yourToken",
"_blank");
```

2.3. Viewer window reference and post message commands

To receive Viewer tab window reference or update the reference which was received on ***window.open*** you can use following function with ***"MedDreamViewer"*** tab name:

```
const windowReference = window.open("", "MedDreamViewer");
```

To post messages use following command:

```
const messageData = {actionType, actionData};
windowReference.postMessage(messageData, targetDomainURL);
```

- **messageData** - Data object which contains command ***actionType*** and ***actionData***.

actionType - Unique constant by which action command is selected.
actionData - Data object that is needed for selected action command.

- **targetDomainURL** - Target domain URL address to which action command is being sent.

2.4. Communication service ready event

When the MedDream Viewer tab is opened and it finishes loading, the ***"COMMUNICATION_SERVICE_READY"*** event is dispatched to the parent page. This event callback can be used to ensure that Viewer is ready to accept action commands. To catch ***"COMMUNICATION_SERVICE_READY"*** event follow the example below:

```
const onMessageReceived = ({data, origin}) => {
  if (origin === TargetDomainAddress
    && data.actionType === "COMMUNICATION_SERVICE_READY"
  ) {
    console.log(data.actionData)
  }
};
window.addEventListener("message", onMessageReceived);
```

3. Available action commands

3.1. For study integration

3.1.1. Open study

```
const actionType = "OPEN_STUDY";
const actionData = {
  study: "example-study-uid"
};
windowReference.postMessage({actionType, actionData}, targetDomainURL);
```

- **actionType** - Command action type is ***"OPEN_STUDY"***.
- **actionData** - Command data object has a ***study*** parameter.
  study - Unique study uid which has to be opened.

3.1.2. Open studies

```
const actionType = "OPEN_STUDIES";
const actionData = {
  studies: [
    "example-study-uid-1",
    "example-study-uid-2",
```

```
    "example-study-uid-3"
  ]
};
windowReference.postMessage({actionType, actionData}, targetDomainURL);
```

- **actionType** - Command action type is ***"OPEN_STUDIES"***.
- **actionData** - Command data object has ***studies*** array.
  studies - Array of unique study uid's to open.

3.1.3. Replace studies

```
const actionType = "REPLACE_STUDIES";
const actionData = {
  studies: [
    "example-study-uid-1",
    "example-study-uid-2"
  ]
};
windowReference.postMessage({actionType, actionData}, targetDomainURL);
```

- **actionType** - Command action type is ***"REPLACE_STUDIES"***.
- **actionData** - Command data object has ***studies*** array.
  studies - Array of unique study uid's to open.

3.1.4. Preload studies

```
const actionType = "PRELOAD_STUDIES";
const actionData = {
  studies: [
    "example-study-uid-1",
    "example-study-uid-2"
  ]
};
windowReference.postMessage({actionType, actionData}, targetDomainURL);
```

- **actionType** - Command action type is ***"PRELOAD_STUDIES"***.
- **actionData** - Command data object has ***studies*** array.
  studies - Array of unique study uid's to open.

3.1.5. Cache studies

```
const actionType = "CACHE_STUDIES";
const actionData = {
  studies: [
    {
      studyUid: "example-study-uid-1",
      storageId: "example-storage-uid-1"
```

```
    },
    {
      studyUid: "example-study-uid-2",
      storageId: "example-storage-uid-2"
    }
  ]
};
windowReference.postMessage({actionType, actionData}, targetDomainURL);
```

- **actionType** - Command action type is ***"CACHE_STUDIES"***.
- **actionData** - Command data object has ***studies*** array. Each item in the ***studies*** array is a study object with ***studyUid*** and ***storageId*** parameters.
  studyUid - Unique study uid which has to be opened.
  storageId - Storage id in which study can be found.

3.1.6. Close studies

```
const actionType = "CLOSE_STUDIES";
const actionData = {
  studies: [
    {
      studyUid: "example-study-uid-1",
      storageId: "example-storage-uid-1"
    },
    {
      studyUid: "example-study-uid-2",
      storageId: "example-storage-uid-2"
    }
  ]
};
windowReference.postMessage({actionType, actionData}, targetDomainURL);
```

- **actionType** - Command action type is ***"CLOSE_STUDIES"***.
- **actionData** - Command data object has ***studies*** array. Each item in the ***studies*** array is a study object with ***studyUid*** and ***storageId*** parameters.
  studyUid - Unique study uid which has to be opened.
  storageId - Storage id in which study can be found.

3.2. For token integration

3.2.1. Open studies with token

```
const actionType = "OPEN_STUDIES_WITH_TOKEN";
const actionData = {
  token: "example-token"
```

```
};
windowReference.postMessage({actionType, actionData}, targetDomainURL);
```

- **actionType** - Command action type is ***"OPEN_STUDIES_WITH_TOKEN"***.
- **actionData** - Command data object has a ***token*** parameter.
  token - Unique token which contains studies information.

3.2.2. Replace studies with token

```
const actionType = "REPLACE_STUDIES_WITH_TOKEN";
const actionData = {
  token: "example-token"
};
windowReference.postMessage({actionType, actionData}, targetDomainURL);
```

- **actionType** - Command action type is ***"REPLACE_STUDIES_WITH_TOKEN"***.
- **actionData** - Command data object has a ***token*** parameter.
  token - Unique token which contains studies information.

3.2.3. Preload studies with token

```
const actionType = "PRELOAD_STUDIES_WITH_TOKEN";
const actionData = {
  token: "example-token"
};
windowReference.postMessage({actionType, actionData}, targetDomainURL);
```

- **actionType** - Command action type is ***"PRELOAD_STUDIES_WITH_TOKEN"***.
- **actionData** - Command data object has a ***token*** parameter.
  token - Unique token which contains studies information.

3.2.4. Cache studies with token

```
const actionType = "CACHE_STUDIES_WITH_TOKEN";
const actionData = {
  token: "example-token"
};
windowReference.postMessage({actionType, actionData}, targetDomainURL);
```

- **actionType** - Command action type is ***"CACHE_STUDIES_WITH_TOKEN"***.
- **actionData** - Command data object has a ***token*** parameter.
  token - Unique token which contains studies information.

3.2.5. Close studies with token

```
const actionType = "CLOSE_STUDIES_WITH_TOKEN";
const actionData = {
```

```
  token: "example-token"
};
windowReference.postMessage({actionType, actionData}, targetDomainURL);
```

- **actionType** - Command action type is ***"CLOSE_STUDIES_WITH_TOKEN"***.
- **actionData** - Command data object has a ***token*** parameter.
  token - Unique token which contains studies information.

3.3. Other action commands

3.3.1. Cache all studies

```
const actionType = "CACHE_ALL_STUDIES";
windowReference.postMessage({actionType}, targetDomainURL);
```

- **actionType** - Command action type is ***"CACHE_ALL_STUDIES"***.

3.3.2. Close all studies

```
const actionType = "CLOSE_ALL_STUDIES";
windowReference.postMessage({actionType}, targetDomainURL);
```

- **actionType** - Command action type is ***"CLOSE_ALL_STUDIES"***.

3.3.3. Set layout

```
const actionType = "SET_LAYOUT";
const actionData = {
  columns: 2,
  rows: 1
};
windowReference.postMessage({actionType, actionData}, targetDomainURL);
```

- **actionType** - Command action type is ***"SET_LAYOUT"***.
- **actionData** - Command data object has ***columns*** and ***rows*** parameters.
  columns - Number of columns in layout.
  rows - Number of rows in layout.

3.3.4. Open instance

```
const actionType = "OPEN_INSTANCE";
const actionData = {
  instanceUid: "example-instance-uid",
  viewportColumn: 2,
  viewportRow: 1,
  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
  }
};
windowReference.postMessage({actionType, actionData}, targetDomainURL);
```

- **actionType** - Command action type is ***"OPEN_INSTANCE"***.
- **actionData** - Command data object has ***instanceUid***, ***viewportColumn***, ***viewportRow*** and ***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.

3.3.4.1. Available viewport actions:

- **windowing** - Windowing level. Available options: ***"DEFAULT"***, ***"AUTO"***, ***"CUSTOM"***. If ***"CUSTOM"*** windowing is selected, ***customWindowing*** parameter has to be defined in the ***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** - Vertical 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"***.

3.3.5. Export instance

```
const actionType = "EXPORT_INSTANCE";
const actionData = {
  viewportColumn: 2,
  viewportRow: 1,
```

```
};
windowReference.postMessage({actionType, actionData}, targetDomainURL);
```

- **actionType** - Command action type is ***"EXPORT_INSTANCE"***.
- **actionData** - Command data object has ***viewportColumn*** and ***viewportRow*** parameters.
  viewportColumn (optional) - Column number of desired viewport.
  viewportRow (optional) - Row number of desired viewport.

If ***viewportColumn*** and ***viewportRow*** are not provided, then the currently active viewport container instance is exported.

3.3.6. Update segmentation tool permissions

```
const actionType = "UPDATE_SEGMENTATION_TOOL_PERMISSIONS";
const actionData = {
  permissions: {
    boundingBoxView: true,
    boundingBox2dEdit: true,
    boundingBox3dEdit: true,
    boundingBoxInfo: false,
    freeDrawView: true,
    freeDrawEdit: true,
    smartPaintView: true,
    smartPaint2dEdit: true,
    smartPaint3dEdit: true,
    smartPaintInfo: false
  }
};
windowReference.postMessage({actionType, actionData}, targetDomainURL);
```

- **actionType** - Command action type is ***"UPDATE_SEGMENTATION_TOOL_PERMISSIONS"***.
- **actionData** - Command data object has ***permissions*** object. ***permissions*** object can contain possible permissions: ***boundingBoxView***, ***boundingBox2dEdit***, ***boundingBox3dEdit***, ***boundingBoxInfo***, ***freeDrawView***, ***freeDrawEdit***, ***smartPaintView***, ***smartPaint2dEdit***, ***smartPaint3dEdit***, ***smartPaintInfo***.
  boundingBoxView (optional) - Allows to see the bounding box tab.
  boundingBox2dEdit (optional) - Allows to edit 2d bounding box data.
  boundingBox3dEdit (optional) - Allows to edit 3d bounding box data.
  boundingBoxInfo (optional) - Allows to use of the bounding box information button and panel.
  freeDrawView (optional) - Allows to see the free draw tab.
  freeDrawEdit (optional) - Allows to edit free draw data.
  smartPaintView (optional) - Allows to see the smart paint tab.
  smartPaint2dEdit (optional) - Allows to edit 2d smart paint data.
  smartPaint3dEdit (optional) - Allows to edit 3d smart paint data.

smartPaintInfo (optional) - Allows to use of the smart paint information button and panel.

Permission receives default value as ***false*** if it is not provided in the ***permissions*** object.

3.3.7. Subscribe event

```
const actionType = "SUBSCRIBE_EVENT";
const actionData = {
  eventType: "EVENT_TYPE" //One of supported event types
};
windowReference.postMessage({actionType, actionData}, targetDomainURL);
```

- **actionType** - Command action type is ***"SUBSCRIBE_EVENT"***.
- **actionData** - Command data object contains ***eventType*** parameter.
  eventType - Event type which has to be subscribed.

Supported event types:

- ***"STUDY_LOADED"*** - Event returns study information after a new study is loaded to the viewer.
- ***"ANNOTATIONS_SAVED"*** - Event returns annotations information after annotations are saved.

To catch MedDream Viewer event callback, you have to register a ***“message”*** event listener, check if the message is from the Viewer tab and confirm if ***actionType*** matches a supported event type. If everything is correct, then you can take callback ***actionData*** object and use it in your use case. Usage example:

```
const onMessageReceived = ({data, origin}) => {
  if (origin === TargetDomainAddress) {
    if (data.actionType === "STUDY_LOADED") {
       console.log(data.actionData); //Do something after study is loaded
    }
    if (data.actionType === "ANNOTATIONS_SAVED") {
      console.log(data.actionData); //Do something after annotations are saved
    }
  }
};
window.addEventListener("message", onMessageReceived);
```

3.3.8. Unsubscribe event

```
const actionType = "UNSUBSCRIBE_EVENT";
const actionData = {
  eventType: "EVENT_TYPE" //One of supported event types
};
windowReference.postMessage({actionType, actionData}, targetDomainURL);
```

- **actionType** - Command action type is ***"UNSUBSCRIBE_EVENT"***.
- **actionData** - Command data object has ***eventType*** parameter.

eventType - Event type constant which has to be unsubscribed.

Supported event types:

- ***"STUDY_LOADED"*** - Event returns study information after a new study is loaded to the viewer.
- ***"ANNOTATIONS_SAVED"*** - Event returns annotations information after annotations are saved.

3.3.9. Get opened studies

```
const actionType = "GET_OPENED_STUDIES";
windowReference.postMessage({actionType}, targetDomainURL);
```

- **actionType** - Command action type is ***"GET_OPENED_STUDIES"***.

To catch a MedDream Viewer callback, you have to register a ***“message”*** event listener, check if the message is from the Viewer tab and confirm if actionType matches to ***"GET_OPENED_STUDIES"***. If everything is correct, then you can take studies data from an ***actionData*** object and use it in your use case. Usage example:

```
const onMessageReceived = ({data, origin}) => {
  if (origin === TargetDomainAddress && data.actionType === "GET_OPENED_STUDIES") {
    console.log(data.actionData)
  }
};
window.addEventListener("message", onMessageReceived);
```

MedDream Viewer ***"GET_OPENED_STUDIES"*** command callback ***data*** object example:

```
data = {
  actionType = "GET_OPENED_STUDIES",
  actionData: {
    studies: [
      {
        studyUid: "example-study-uid-1",
        storageId: "example-storage-uid-1"
      },
      {
        studyUid: "example-study-uid-2",
        storageId: "example-storage-uid-2"
      }
    ]
  }
};
```

3.3.10. Get viewport data

```
const actionType = "GET_VIEWPORT_DATA";
windowReference.postMessage({actionType}, targetDomainURL);
```

- **actionType** - Command action type is ***"GET_VIEWPORT_DATA"***.

To catch a MedDream Viewer callback, you have to register a ***“message”*** event listener, check if the message is from the Viewer tab and confirm if actionType matches to ***"GET_VIEWPORT_DATA"***. If everything is correct, then you can take viewport data from an ***actionData*** object and use it in your use case. Usage example:

```
const onMessageReceived = ({data, origin}) => {
  if (origin === TargetDomainAddress && data.actionType === "GET_VIEWPORT_DATA") {
    console.log(data.actionData)
  }
};
window.addEventListener("message", onMessageReceived);
```

MedDream Viewer ***"GET_VIEWPORT_DATA"*** command callback ***data*** object example:

```
data = {
  actionType = "GET_VIEWPORT_DATA",
  actionData: {
    0020000D: "study-uid",
    0070005A: [{...}],
    00080016: "sop-class-uid",
    00080060: "PR",
    00081115: [{...}],
    00283110: [{...}],
    00700041: "N",
    00700042: 0,
    00700080: "Presentation state",
    00700081: "",
    00700082: "20221219",
    00700083: "",
    00700084: "",
    20500020: "IDENTITY",
    image: "..."
    patientId: "patient-id"
  }
};
```

3.3.11. Get snapshot

```
const actionType = "GET_SNAPSHOT";
windowReference.postMessage({actionType}, targetDomainURL);
```

- **actionType** - Command action type is ***"GET_SNAPSHOT"***.

To catch the MedDream Viewer callback, you have to register a ***“message”*** event listener, check if the message is from the Viewer tab and confirm if actionType matches to ***"GET_SNAPSHOT"***. If everything is correct, then you can take the generated layout and viewports snapshot from an ***actionData*** object and use it in your use case. Usage example:

```
const onMessageReceived = ({data, origin}) => {
```

```
  if (origin === TargetDomainAddress && data.actionType === "GET_SNAPSHOT") {
    console.log(data.actionData)
  }
};
window.addEventListener("message", onMessageReceived);
```

Received snapshot object is used for ***"SET_SNAPSHOT"*** command to restore saved studies, layout and viewports information to MedDream Viewer.

3.3.12. Set snapshot

```
const actionType = "SET_SNAPSHOT";
const actionData = {
  snapshot: {        //Snapshot object received from "GET_SNAPSHOT" command
    ...
  }
};
windowReference.postMessage({actionType, actionData}, targetDomainURL);
```

- **actionType** - Command action type is ***"SET_SNAPSHOT"***.
- **actionData** - Command data object has a ***snapshot*** object.
  snapshot - Snapshot object has studies, layout and viewports information. This object is received from the ***"GET_SNAPSHOT"*** command and used to restore saved studies, layout and viewports information to MedDream Viewer.

4. Documentation updates

4.1. Update - 2021-01-22

- 3.3.9. Get opened studies - Added ***"GET_OPENED_STUDIES"*** command callback ***data*** object example.

4.2. Update - 2021-03-05

- 3.1.5. Cache studies - Renamed old ***"Preload studies"*** action command to ***"Cache studies"***. Update action type from ***"PRELOAD_STUDIES"*** to ***"CACHE_STUDIES"*** in your integration;
- 3.2.4. Cache studies with token - Renamed old ***"Preload studies with token"*** action command to ***"Cache studies with token"***. Update action type from ***"PRELOAD_STUDIES_WITH_TOKEN"*** to ***"CACHE_STUDIES_WITH_TOKEN"*** in your integration;
- 3.3.1. Cache all studies - Renamed old ***"Preload all studies"*** action command to ***"Cache all studies"***. Update action type from ***"PRELOAD_ALL_STUDIES"*** to ***"CACHE_ALL_STUDIES"*** in your integration;
- 3.1.4. Preload studies - New action command to preload studies in viewer memory. Preloaded studies are not visible in the viewer until the ***"Open study/studies"*** action command is performed;
- 3.2.3. Preload studies with token - New action command to preload studies with token in viewer memory. Preloaded studies are not visible in the viewer until the ***"Open studies with token"*** action command is performed.

4.3. Update - 2021-09-29

- 3.3.6. Update segmentation tool permissions - Added new ***"UPDATE_SEGMENTATION_TOOL_PERMISSIONS"*** command to control segmentation tool permissions.
- 3.3.7. Subscribe event - Added new ***"SUBSCRIBE_EVENT"*** command which can be used for subscribing to supported events.
- 3.3.8. Unsubscribe event - Added new ***"UNSUBSCRIBE_EVENT"*** command to unsubscribe from subscribed event.

4.4. Update - 2021-11-11

- 3.3.6. Update segmentation tool permissions - Updated segmentation permission ***boundingBoxEdit*** to ***boundingBox2dEdit*** and ***boundingBox3dEdit*** for 2d and 3d bounding box permissions control.

4.5. Update - 2021-12-15

- 3.3.6. Update segmentation tool permissions - Added new smart paint segmentation tool permissions: ***smartPaintView***, ***smartPaint2dEdit***, ***smartPaint3dEdit***, ***smartPaintInfo***.
- 3.3.7. Subscribe event - Updated ***"ANNOTATION_SAVED"*** event name to ***"ANNOTATIONS_SAVED".***

4.6. Update - 2022-02-01

- 2. Set up - Moved old set up information to ***"2.1. Configuration"***, ***"2.2. Open MedDream Viewer"***, ***"2.3. Viewer window reference and post message commands"*** sub sections.
- 2.4. Communication service ready event - Added information on how to subscribe to the ***"COMMUNICATION_SERVICE_READY"*** event.

4.7. Update - 2022-03-14

- 3.3.11. Get snapshot - Added new ***"GET_SNAPSHOT"*** command which can be used to receive snapshot of studies, layout and viewports information from MedDream Viewer.
- 3.3.12. Set snapshot - Added new ***"SET_SNAPSHOT"*** to restore received snapshot object back to the MedDream Viewer.

4.8. Update - 2022-11-08

- 3.3.7. Subscribe event - Added information about new ***"STUDY_LOADED"*** event.
- 3.3.8. Unsubscribe event - Added information about supported event types.

4.9. Update - 2022-12-19

- 3.3.10. Get viewport data - Added new ***"GET_VIEWPORT_DATA"*** command which can be used to receive active viewport data from MedDream Viewer.

4.10. Update - 2023-03-07

- 3.3.6. Update segmentation tool permissions - Removed information about not used ***freeDrawInfo*** permission.