



**Project acronym: CS3MESH4EOSC**

**Deliverable D3.3: Application-specific extensions integrated with the protocol**

Contractual delivery date	30/09/2021
Actual delivery date	21/10/2021
Grant Agreement no.	863353
Work Package	WP3
Nature of Deliverable	R (Report)
Dissemination Level	PU (Public)
Lead Partner	CERN
Document ID	CS3MESH4EOSC-20-014
Authors	Hugo Gonzalez Labrador (CERN), Daniel Müller (WWU), Holger Angement (WWU), Giuseppe Lo Presti (CERN), Guido Aben (AARNet), Pedro Ferreira (CERN), Marcin Sieprawski (AILLERON), David Antos (CESNET)



**Disclaimer:**

The document reflects only the authors' view and the European Commission is not responsible for any use that may be made of the information it contains.



*This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 863353*

# Table of Contents

Table of Contents.....	2
Table of figures .....	2
1 Introduction .....	3
2 Overview and Use Cases .....	3
2.1 Data Science Environments .....	3
2.2 Open Data Systems.....	3
2.3 Collaborative Documents .....	4
2.4 On-demand Data Transfers .....	4
3 Data Science Environments .....	5
3.1 JupyterLab extension - integration with CS3 API.....	5
3.2 Concurrent editing of Notebooks: locking shared files .....	7
3.3 Locking support in CS3 APIs - locking/merging of notebooks is a use case for this .....	8
3.4 Locks in the Jupyter plug-in – implementation (integration with IOP) .....	8
4 Open Data Systems.....	9
5 Collaborative Documents .....	10
6 On-demand Data Transfers .....	11
7 Conclusion.....	13

## Table of figures

Figure 1. JupyterLab – CS3 API integration back-end component architecture .....	5
Figure 2. Structure of the lock object .....	9
Figure 3. Flow of locking in notebooks .....	9
Figure 4. Direct data transfer; pull model work-flow .....	13

# 1 Introduction

This document describes all application-specific extensions integrated with the protocols and APIs developed for the Science Mesh project. All these extensions stem from work done in Work Package 4, *Users and Applications*. Our approach is based on:

1. extending protocols already in use by the CS3 community (e.g., OCM, CS3APIs), and
2. adopting other existing standard tools and protocols wherever appropriate (e.g. WOPI, Rclone, ...).

This document first gives an overview of all applications and their respective use cases of the various application-specific extensions. The following chapters will then describe each extension in greater detail.

## 2 Overview and Use Cases

In this section, we present an overview of the various applications and respective use cases for which specific extensions are integrated with the protocols.

### 2.1 Data Science Environments

The distributed Data Science Environments use case aims to facilitate collaborative research and enable sharing of computational tools, algorithms, and resources across the ScienceMesh. This is to allow users to access remote execution environments to replay (and modify) analysis algorithms without the need to set up accounts upfront in the remote services. Interconnection of a web-based Jupyter environment with the web-based EFSS environment is ideally suited for this task.

The integration between Jupyter and EFSS services is achieved by a JupyterLab extension which uses CS3APIs to communicate with local EFSS and, indirectly, via OCM, with remote sites/environments. In this way access to federated resources is provided directly from the Jupyter service, thus enabling researchers to perform data science tasks in a federated cloud. This allows a user to move back-and-forth between a local file system and a federated Science Mesh within one cohesive platform shared by users and with easy access to the science tools.

The Data Science Environments use case will be further detailed in Section 3 of this document.

### 2.2 Open Data Systems

Open Data Systems help the Science Mesh users to handle structured research-related data and use it for activities typically associated with it (archive, deposit and publish). The following interactions

should be possible:

1. Users should be able to create structured datasets out of unstructured data present in their home mesh node or about to be ingested into it – that is, delimit what data and in what ordered format (typically directory trees) is included in the dataset to be defined. The project will leverage the RO-Crate specification and its structured data format.
2. Users should be able to manipulate the dataset’s metadata. For this purpose, the Describo<sup>1</sup> tool provided by UTS will be a starting point, closely collaborating with the authors to explore its capabilities and integration.
3. Users should be able to augment the structured dataset with appropriate metadata, either user-generated or harvested from environment data. Here, too, Describo acts as the design pattern.
4. Users should be able to perform scientific workflow actions (publish, archive, etc) based on RO-Crates<sup>2</sup>, either those already present in the system or RO-Crates that must be generated on demand, to fulfil the task. ScieboRDS<sup>3</sup> is our design example here, and WP4.2 will work closely with the ScieboRDS team to maximise synergies. Some of these workflows will start inside the ScieboRDS application environment (e.g., “a scientist wants to publish a paper with a certain dataset attached”), hand over to the Describo environment (“select the relevant data to be crated up”) and then hand back to ScieboRDS (“here’s the crated and metadata-augmented data the scientist wanted to publish”).

## 2.3 Collaborative Documents

Collaboratively editing a document, usually on a web browser, requires the application to be able to synchronize the document data to and from the underlying storage. This means that the application needs to be able to “read” and “write” the document no matter whether it resides on the user’s sync and share system or if it has only been shared with them by another user.

While such an application can achieve this by directly using the CS3APIs, another simpler approach is based on the WOPI protocol (cf. 4) as it exposes higher-level endpoints to manage file locking and document collaboration.

## 2.4 On-demand Data Transfers

On-demand data transfer fulfils the use-case where it is not possible to extend processing capabilities to remote sites. Instead, the data typically in the TB size range, needs to be transferred close to where the processing capabilities are. Two modes of operation for this will be made available within the

---

<sup>1</sup> <https://ereseach.uts.edu.au/tools/describo/>

<sup>2</sup> <https://ereseach.uts.edu.au/tools/describo/>

<sup>3</sup> <https://www.research-data-services.org/de/>

Science Mesh: 1) transfers initiated and handled directly (point-to-point) between EFSS services; 2) transfers initiated by EFSS service but offloaded to a secondary service (such as [FTS](#)). As a primary transfer tool candidate for direct file transfer, [Rclone](#)<sup>4</sup> is used.

### 3 Data Science Environments

For distributed data science environments we develop JupyterLab extension (cs3api4lab), integrating with IOP – so that inside the JupyterLab environment we can provide file browsing and additional share and collaboration functionalities for notebooks and resources in a distributed cloud.

#### 3.1 JupyterLab extension - integration with CS3 API

The back-end part of the JupyterLab extension is implemented by replacing the ContentsManager and Checkpoints components of the JupyterLab as well as providing REST endpoints for integration with the front-end. They expose APIs for content operations, checkpoints operations and share operations. The ScienceMesh's IOPS is interfaced by gRPC (the CS3 APIs). The Figure xx below depicts the back-end architecture.



Figure 1. JupyterLab – CS3 API integration back-end component architecture

The back-end provides the following REST endpoints for integration with the front-end:

API for content operations:

- **/api/contents/{path} (GET)** – This endpoint returns a model for a file, notebook or directory. The directory model contains a list of models, the notebook model contains a notebook data in JSON format, the file model contains file data in a text or base64 format. This endpoint uses the CS3 APIs' methods: Authenticate, ListContainer, InitiateFileDownload
- **/api/contents/{path} (PUT)** – Saves a file or a notebook in the specified file path. For the notebook model, the content key is saved in a JSON format. For the file format, the key context is kept unchanged. If the JSON body contains `{"copy_from":"[path/to/]OtherNotebook.ipynb" }`, the file copy operation is performed. This endpoint uses the CS3 APIs' methods: Authenticate, ListContainer, CreateContainer, InitiateFileUpload, InitiateFileDownload
- **/api/contents/{path} (PATCH)** - Renames a file, a notebook or a directory without re-

---

<sup>4</sup> <https://rclone.org/>

uploading content. This endpoint uses the CS3APIs' methods: Authenticate, ListContainer, Move

- **/api/contents/{path} (POST)** - Creates a new file, notebook or directory in the specified path. The server always decides on the file or directory name. This endpoint uses the CS3APIs' methods: Authenticate, ListContainer, CreateContainer, InitiateFileUpload, InitiateFileDownload
- **/api/contents/{path} (DELETE)** - Deletes a file or directory in the given path. This endpoint uses the CS3APIs' methods: Authenticate, Delete

API for checkpoints operations:

- **/api/contents/{path}/checkpoints (GET)** - Lists checkpoints for a file.
- **/api/contents/{path}/checkpoints (POST)** - Creates a new checkpoint for a file.
- **/api/contents/{path}/checkpoints/{checkpoint\_id} (POST)** - Restores a file from a checkpoint.
- **/api/contents/{path}/checkpoints/{checkpoint\_id} (DELETE)** - Deletes a checkpoint for a given file.

API for share operations:

- **/api/cs3/shares (POST)** - Creates a share. This endpoint uses the CS3APIs' methods: Authenticate, CreateShare.
- **/api/cs3/shares (DELETE)** - Deletes a share. This endpoint uses the CS3APIs' methods: Authenticate, RemoveShare.
- **/api/cs3/shares (PUT)** - Updates a share. This endpoint uses the CS3APIs' methods: Authenticate, UpdateShare
- **/api/cs3/shares/list (GET)** - Gets all given shares. This endpoint uses the CS3APIs' methods: Authenticate, ListShares
- **/api/cs3/shares/received (GET)** - Gets all received shares. This endpoint uses the CS3APIs' methods: Authenticate, ListReceivedShares.
- **/api/cs3/shares/received (PUT)** - Updates received share. This endpoint uses the CS3APIs' methods: Authenticate, UpdateReceivedShare
- **/api/cs3/shares/file (GET)** - Returns grantees for file. This endpoint uses the CS3APIs' methods: Authenticate, ListShares

## 3.2 Concurrent editing of Notebooks: locking shared files

Collaborative Data Science workflows require that apart from sharing notebooks it is possible that for a group of scientists to work concurrently on the same shared notebook.

While real-time collaborative editing of notebooks, similar to collaborative editing of documents in T4.3, was considered as a solution for this, an alternative solution was chosen, to allow users to open notebooks concurrently without causing conflicts. This solution is based on locking the notebooks, working on separate copies in case the notebook is locked, and merging the inputs. The main reasons for this design choice are:

- Domain analysis showed that locking/merging feature is more appropriate to support typical workflows for collaboration on notebooks
- This feature is of interest of JRC and CERN – the scientific use cases for distributed data science environments, and Ailleron - Software Mind (for business related use case)
- Jupyter notebooks (unlike documents) allow code executions. The locking/merging feature is more suitable for this software development perspective, offers better predictability of results and avoiding running “half baked” cells edited by another collaborator. While real-time editing of notebooks is interesting, it would require some more analysis of those scenarios.
- The current status of collaborative editing of notebooks is still uncertain, with a considerable effort still needed. It is not available currently in JupyterLab, and the existing prototypes we evaluated have a low level of readiness.

The locking/merging feature takes a similar approach to the editing of office files. This means, we would lock a notebook if someone else is editing, which would only allow another person to open it in read-only mode. This would prevent the current Jupyter behavior of complaining that a file has changed on the disk and the possible loss of data.

But, we can still allow other's to contribute. We can ask the user if he just wants to open (read -only) or if he wants to create a copy and merge it after. This copy should have a specific name, following some convention, that would allow the system to detect it and suggest a merge at a later stage.

Opening a file should provide the following information:

- Is the notebook locked or not, and by whom?;
- Do we have a copy of the notebook that was initiated by us?

The copies of locked notebooks will be stored locally, but locking mechanism has to be connected to the shared file in IOP. This means that we need to create methods for locking, unlocking, getLockStatus.

The UI also needed to be changed to support this new concept, and the workflow for working with

locked notebooks: in case a notebook is considered locked, it needs to ask the user to open in read-only or create a copy; It also needs to suggest the merge of the changes.

### 3.3 Locking support in CS3 APIs - locking/merging of notebooks is a use case for this

In CS3 APIs currently, there is no locking support. For Reva/IOP there is an active ticket for “storage: locking support” (<https://github.com/cs3org/cs3apis/issues/6>), stating that the CS3 APIs should allow implementing locking functionality for e.g. webdav or s3 – but the rationale and priority of that feature is still discussed/analysed. The locking/merging feature in CS3 Jupyter plugin (cs3api4lab) is listed in this ticket as one of the use cases (“Simple approach for concurrent opening of Notebooks” sciencemesh/cs3api4lab#11).

Some additional aspects need to be considered on implementing the locking support in CS3 API (atomic operations, using this by various systems etc.) – to be analysed in the design of this feature, combining inputs from all use cases.

### 3.4 Locks in the Jupyter plug-in – implementation (integration with IOP)

As locking mechanism is not supported in IOP, locks of shared files are implemented in Jupyter plugin using CS3 API metadata support: ArbitraryMetadata field in metadata is used to store information about locks:

- Only one lock per file can be present at a time
- Whenever a shared file is opened or saved the plugin checks for the existence of a lock
  - If there is no lock, the file opens/saves, and a lock is created (by updating ArbitraryMetadata of the file)
  - If there is a lock, the user is given an option to either work on a copy or open the file in a preview mode (when opening) or an error is shown (when saving)
  - We also check if the lock is ours or if it expired, in both cases the lock is ignored
- Please note, that there is an important assumption, that no other system uses ArbitraryMetadata field for files shared for use in the Jupyter plugin. Overriding this information by another system can cause loss/overriding of some inputs in collaborative editing of notebooks.

The structure of the lock object, stored in ArbitraryMetadata field, is the following:



```

"username": str
r
"idp": str
"opaque_id": str
"updated": float
at
"created": float
at

```

Figure 2. Structure of the lock object

The diagram below presents the flow for locking shared notebooks in Jupyter extension:

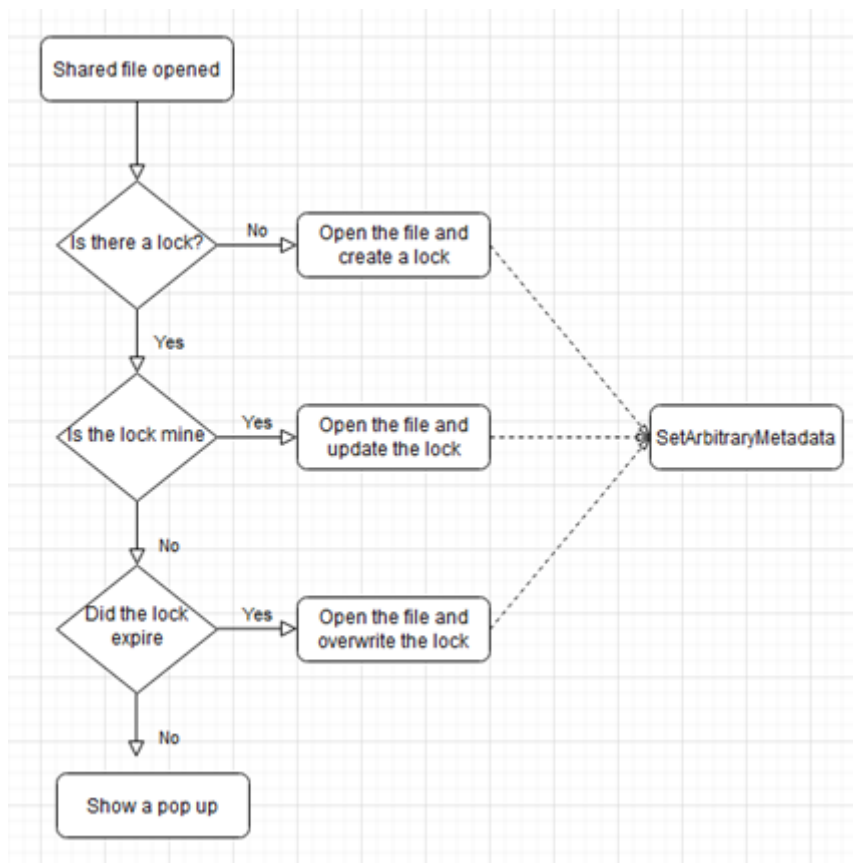


Figure 3. Flow of locking in notebooks

## 4 Open Data Systems

Open data systems necessarily depend upon standards for data description and packaging to be able to share / move data packages between services. For this reason, the Research Object Crate (RO-Crate) standard has been selected as the packaging and description format for the Science Mesh.

By adopting the Describo tool and integrating it into the Science Mesh as a supported, core application, all mesh users will be able to describe their data hosted on the mesh so that it can later be transferred to archival and repository systems.

Describo enables the creation of RO-Crates by writing into a folder a ro-crate-metadata.json file that describes the content of the folder. Users can choose what level of description they require for their work from defining a basic name for the data all the way to describing individual files and their properties.

Describo interacts with the mesh in 3 main ways:

- File list operations giving a target folder
- File copy from the mesh - specifically, copying a ROcrate file in a target folder back to its internal cache for updates
- File copy to the mesh - specifically, synchronising the local cached version of the RO-crate file back to the target folder.

Design discussions are on-going about the best way to integrate the Describo tool with Reva (using the CS3APIS or the Rclone support) to interact with backend storage systems.

## 5 Collaborative Documents

Collabora is a powerful online office suite that supports all major document, spreadsheet and presentation file formats, which can be integrated in your own infrastructure. Key features are collaborative editing and good office file format support. Key use cases for office files are especially the creation of papers, proposals and reports for joint projects. These are often written together with peers from different institutions.

CodiMD is an open-source, web-based, self-hosted, collaborative markdown editor. It can be used to easily collaborate on notes, graphs and presentations in real-time. CodiMD is especially useful as a note-taking tool, for example during on- and offline meetings. Also, writing documentations and specifications is a major use case.

When it comes to the actual back-end implementation, most of the integration mechanics developed within the T4.3 task have been built on top of the WOPI protocol (Web Application Open Platform Interface Protocol). WOPI is a standard for online collaboration developed by Microsoft for its Office applications, and adopted by Collabora Online as well. Therefore, integration with Collabora Online was straightforward thanks to the WOPI Server package, which allows the IOP to communicate

through that protocol. Since OnlyOffice also integrated WOPI in their latest version, integration can begin soon.

With applications such as CodiMD and EtherPad, which do not support the WOPI protocol natively but have a similar workflow, a slightly different route was taken, which implied creating a WOPI Bridge module, which has in the meantime been integrated into the main WOPI server package. The WOPI Bridge has been developed by CERN for its CERNBox service and adopted and extended for the purposes of CS3MESH4EOSC. Thanks to it, file updates can be triggered in the storage system (e.g. EFSS), in response to notifications received from the online editing application.

We intend to work with the company which develops CodiMD (HackMD) towards possible provision of a native WOPI connector directly on the tool.

The integration work also included the necessary extensions of the Application Registry and the Application Providers logic within the IOP. Now it is possible to register multiple applications, each supporting multiple and possibly overlapping file types, and the site administrator is given the ability to specify which application is the preferred one for each file type. This is also reflected in the web UI, where the preferred application is associated to the default action to open a file, and the context menu is populated with an *“Open with”* item for each registered application.

It’s important to note that the integration approaches use, wherever possible, industry standards such as WOPI. The project supported further development of the technical connector component which enables this capability.

The WOPI server component may be easily re-used to integrate a broader spectrum of applications in the future.

## 6 On-demand Data Transfers

The first use-case that has been implemented is direct (EFSS to EFSS) data transfer. Here an EFSS user wants to transfer data to another user, typically of another EFSS. To initiate the transfer the OCM share mechanism is used, in this case for a specific kind of share, a data transfer type share. The [Rclone](https://rclone.org/)<sup>5</sup> tool is used for the actual data transfers. Two APIs are required for this, OCM and CS3 and the workflow is as follows:

The sender sends a transfer type OCM share request to the receiver that looks like this:

---

<sup>5</sup> <https://rclone.org/>

```

POST /share

  "shareWith": receiving user Id,

  "name": resource name,

  "providerId": provider Id,

  "owner": sending user Id,

  "protocol": {

    "name": "datatx",

    "options": {

      "permissions": permissions,

      "token": token,

    },

  },

  "meshProvider": sending Idp,

}

```

Just like a regular OCM share this request contains all the information needed to set up the data transfer. The distinguishing feature is the protocol name "datatx" which makes it clear to the receiver that this is a data transfer type share as opposed to a regular share. This allows the receiving EFSS to act accordingly.

Once the receiver discovers this share in his list of received OCM shares, he or she can decide to accept it. Next, the EFSS of that user will immediately initiate the transfer through the CS3 data transfer API. This will trigger the transfer tool to start the actual transfer. The transfer is a pull action meaning that the receiving EFSS will pull the data from the sending EFSS. The receiving EFSS is therefore in control of the transfer.

A number of CS3 data transfer functions are available to control the transfer. These functions are implemented in the data transfer service and driver:

- PullTransfer                    - instruct the transfer tool to pull the data
- ListTransfers                - return the list of transfers, optionally filtered
- GetTransferStatus           - return the transfer status
- CancelTransfer              - instruct the transfer tool to cancel the transfer

- RetryTransfer - instruct the transfer tool to restart a transfer

Figure 4 explains the direct data transfer workflow from start (creating the transfer type share) to finish (pulling the data) as currently implemented.

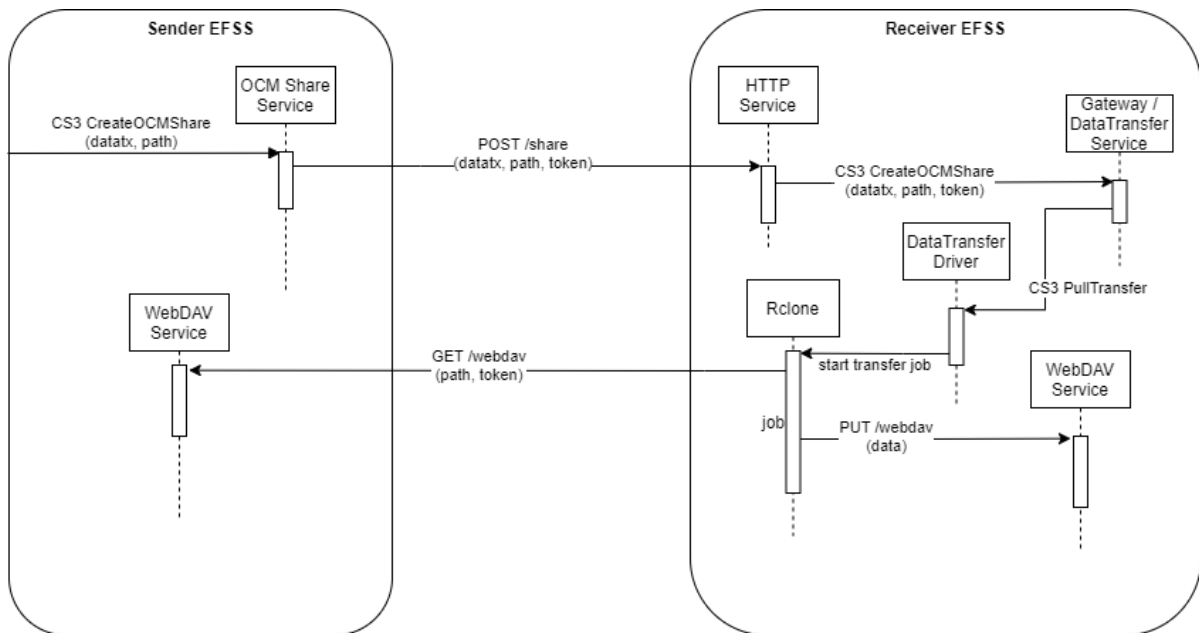


Figure 4. Direct data transfer; pull model work-flow

Rclone uses basic HTTP GET and PUT methods on the WebDAV endpoints of the EFSSs to perform the actual transfer. As WebDAV endpoints are already integral to the EFSSs no additional development was needed on that part.

## 7 Conclusion

This document presents extensions to various APIs and protocols that are necessary for the integration of specific applications into the Science Mesh.

We leverage existing standards as much as possible and introducing new extensions only where needed.