natESM

Sprint Documentation #8

ICON-A/MESSy ComIn integration

Wilton Jaciel Loch[1] , Kerstin Hartung[2], Astrid Kerkweg[3], Bastian Kern[2], Patrick Jöckel[2]

[1] Deutsches Klimarechenzentrum GmbH, Hamburg, Germany
[2] Deutsches Zentrum für Luft- und Raumfahrt, Oberpfaffenhofen, Germany
[3] Forschungszentrum Jülich GmbH, Jülich, Germany

Contact: info@nat-esm.de

Published on 29.04.2024 on https://www.nat-esm.de/services/accepted-sprints

1 Summary

The sprint was intended to integrate the newly developed ICON community interface 'ComIn' into the Modular Earth Submodel System (MESSy) to connect MESSy to ICON-A via ComIn. This integration aimed to facilitate easier adoption of upstream ICON developments and simplified code maintenance through a generalized interface. Given the novelty of ComIn, the developments should also serve as reference for other models.

During the sprint, challenges included dependencies of the existing ICON/MESSy integration on internal ICON structures and subroutines, the lack of a few specific functionalities and data structures in the ComIn interface required for MESSy, and the evolving development state of ComIn. During the sprint, when ComIn was still under development and not yet released, the last two points were closely linked. This means that certain developments in ComIn incorporated functionalities proposed during the ICON/MESSy sprint. The timing of the sprint, which coincided with ComIn's development phase, was intentional. This allowed us to gain early lessons with a complex plugin and provide feedback for ongoing ComIn definition and extension.

The main outcome of the sprint was a functional prototypical MESSy ComIn plugin covering model initialization and some parts of the time loop. Although not complete, it serves as a showcase of successful ComIn utilization in a complex Earth System Model component. Several ComIn developments and design decisions were influenced by sprint requirements, aiding in interface generalization for additional use cases. We produced a detailed guide documenting the sprint to assist other model developers in adopting ComIn.

2 General Information

Start and end date:	10.07.2023 – 10.01.2024
Intended period:	06 months
Responsible RSE:	Wilton Jaciel Loch, DKRZ
Responsible scientist:	Kerstin Hartung, DLR-PA

The Modular Earth Submodel System (MESSy, Jöckel et al., 2005, 2010, 2016) is an integrated software framework providing an infrastructure (data types and methods) for connecting atmospheric models (e.g. ICON, ECHAM, COSMO) with specialized Earth System Model (ESM) components, the so-called submodels (e.g. physical parameterizations, chemistry packages, diagnostics) via generalized interfaces for standardized control and coupling.

The ICON community interface 'ComIn' organizes the inclusion of calls from an external model/software into ICON and sharing of data between the two. ComIn is neither an integrated framework, nor a coupler: the interface controls what, how and when external functions are called within ICON, and how and when data is made available or exchanged.

During the sprint, Comin was not released yet and therefore still subject to frequent changes. After the sprint (end of January 2024), ComIn was made available alongside the ICON open-source release. Currently, there are no known production ESMs using the interface.

MESSy, ICON and ComIn are mainly written in Fortran, with the latter also having C/C++ and Python bindings.

3 Sprint Objectives

Up until now, MESSy was connected to ICON with as few as possible direct modifications of the ICON code (Jöckel et al., 2005; Kern and Jöckel, 2016). Nevertheless, this means that adjustments were required with each update of the ICON version, resulting in a very time-consuming process.

The objective of the sprint was to develop a ComIn plugin of MESSy, referred to as the MESSy ComIn plugin, to facilitate the integration of MESSy with ICON through the ComIn interface. This setup is intended to serve two purposes. Firstly, it serves as a proof of concept for utilizing ICON/ComIn with a complex model component, providing an example and reference for other developers. Secondly, it represents an initial step towards a more efficient method of integrating the production MESSy code with the upstream ICON developments.

4 Procedure and Insights

4.1 Technical Approach / Procedure

The work defined for the sprint was mainly sequential and from the start there was a list of steps to achieve the final goal. This involved three focal points:

1. Removing or replacing the existing ICON dependencies in the MESSy code to get to an operational setup with code and functionality separation, essential for achieving improved integration.
2. Integrating ComIn into the MESSy build system and enabling the creation of a MESSy ComIn plugin.
3. Adding calling sequences of MESSy subroutines to the MESSy ComIn plugin callback functions, to execute the latter via the interface by the host model ICON.

For the third point a list of all the subroutines, or MESSy entry points, which should be included in the callbacks was already provided by the MESSy developers at the beginning of the sprint, turning it into a rather well-defined set of steps. In order to achieve that, however, ComIn needed to already be a part of the build system of MESSy, stating therefore a clear dependency on point 2. Finally, in order for a ComIn plugin to be successfully built with MESSy code, point 1 had to be solved, as otherwise the existing dependencies would prevent compilation of the code.

Given the points above, in the first meetings the sprint team discussed how to solve or work around the ICON dependencies. Since actually replacing or re-implementing these dependencies would have taken a large amount of time, the RSE suggested the solution to pack all existing ICON structures and subroutines, which MESSy depended upon, into the initial version of the new MESSy-ComIn plugin. Although from a functional point of view most of these dependencies would

not be usable due to their reliance on a non-existent initialized ICON state, they allowed a proper preparation of the build system.

The next step was the adaptation of the MESSy-build system to include the ComIn library and to prepare MESSy as a ComIn plugin. This included adding the ICON core as a dependency library for MESSy, including all the relevant ComIn-library subroutines and packing the remaining MESSy-core-library code. The main challenges in this phase were related to the somewhat unorthodox resulting library, because it includes direct dependencies to ICON and is linked to ICON (via ComIn) at the same time, demanding careful consideration of linker tools, functionalities, and options. We want to stress, that this unorthodox approach is only for an intermediate development stage, to enable compilation and linking, whereas the final goal is to separate ICON/ComIn from MESSy/ComIn entirely at configuration and compile times according to the basic ComIn concept.

Once the build infrastructure was in place, the remainder of the time was dedicated to the actual functional inclusion of MESSy entry points into the ComIn plugin. This was organized in a sequential and incremental way, following the previously provided list of entry points and starting with the model initialization. At each step of the process, it was ensured that the current setup could be compiled and executed. During these developments a number of issues arose, which could be resolved during the sprint. Most related to updates required in MESSy, others required careful consideration of callback order and data management in both models, and for some, ComIn had to be expanded. Examples of such challenges were the following.

- The data allocation and management for both ICON and MESSy fields and their optimal combination.
- The bookkeeping of additional metadata in MESSy for ICON variables (including tracers).
- The reception and registration of ICON fields which were not initially available through ComIn, as for example because ICON variables on most vertical axes were filtered out initially.
- Some re-ordering of the calling sequence, which was originally not compatible with the ComIn callbacks.

4.2 General Insights

The first insights came from the build-system adaptation for creating the plugin. As all of the ICON core was also included in the newly developed MESSy/ComIn shared library to support the MESSy-ICON dependencies, a question arose regarding whether the symbols defined in the plugin would conflict with the symbols defined in the host ICON model. The answer was that by default all the symbols defined in a shared library are isolated from the software calling it, and therefore there were no conflicts. Many of these aspects were also found to be configurable according to the linker parameters and linker drivers, allowing for potentially different configurations of linking.

The initial usage of ICON with ComIn brought up a bug in the mtime library, related to the line length limit due to file-name substitutions. In discussion with ComIn developers they noted that this bug had already been reported. The bug was not fixed during the duration of the sprint.

At the initial stages of the sprint, the sprint team discussed how ComIn should be integrated into the MESSy code base. Options were for example as a git subtree or a git submodule inside the MESSy repository, but the sprint team decided to not include ComIn in the MESSy repository. In the end a user will retrieve code from the repositories of both MESSy and ICON, of which the latter already includes ComIn. To avoid ComIn version incompatibility, MESSy is built with ComIn shipped with ICON. Due to this approach, we decided to accept loss of information on matching commits during the co-development of ComIn and the MESSy ComIn plugin. Additionally, during the development even matching commits don't produce a complete working setup. Once the ComIn integration for MESSy is complete, the sprint team will recommend one stable version of

ICON/ComIn to MESSy users. The matching ICON and ComIn commits for each state of MESSy will be included in the MESSy README.md, or a similar file.

5 Results

The main result achieved during the sprint was the successful creation of an initial ComIn plugin for the integration of MESSy with ICON. The current state of the plugin does not comprise the full MESSy execution, but is restricted mainly to the model initialization phase, which is mostly executed and finalized. Its creation sets the stage and provides the required infrastructure for the continuation of the integration with MESSy time-loop subroutines. The initialization phase is likely also the most complex, as MESSy and ICON are strongly synchronized in the existing ICON/MESSy integration.

The dependency on certain ICON functionalities (mainly MPI implementation and metadata not provided by ComIn) led to the creation of a peculiar build configuration, which packs the ICON core library, ComIn and MESSy code all together. This approach allowed us to focus on the actual integration work by avoiding a long process of dependency removal. Eventually the dependencies will of course need to be resolved by adding these functionalities to MESSy. The detailed build configuration is depicted in Figure 1 and might be useful for other models facing restrictive dependencies when creating a ComIn plugin.

Apart from the actual MESSy ComIn plugin created, we gathered valuable information during the development process and through the challenges faced. We have compiled these into a detailed report of the process, which may be of use for other developers interested in the creation of ComIn plugins for their models (see Loch et al., 2024).

Finally, as ComIn was in active development during the sprint, we used several of the requirements of the MESSy ComIn plugin to add new features to ComIn as well as to inform about functionalities, which might be useful to be added to ComIn in the future. This will hopefully allow the consolidation of ComIn by a broader range of applications due to its more generalized and convenient interface.

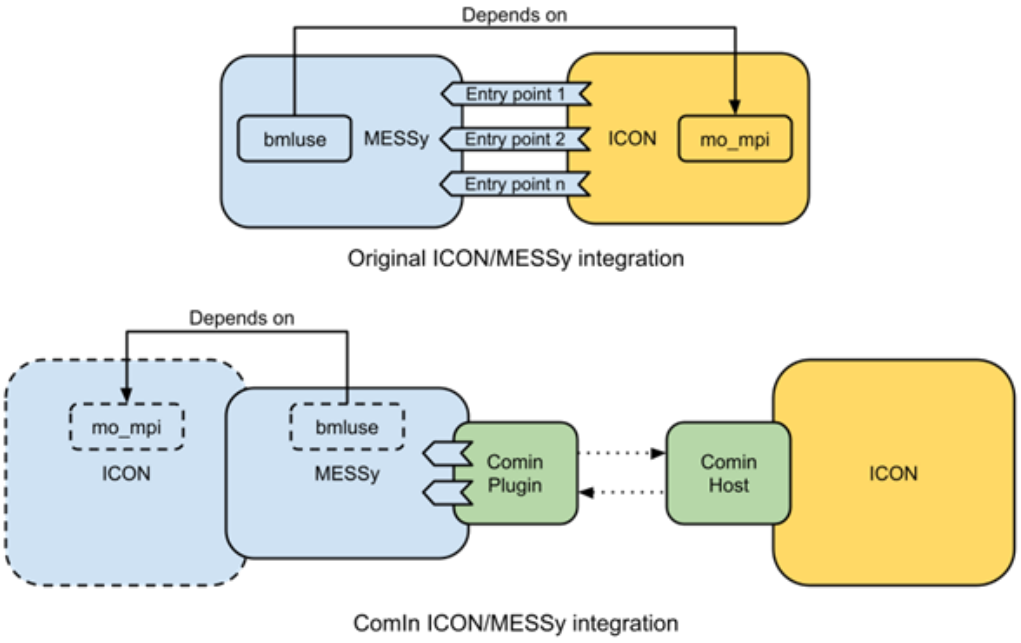

Figure 1: Traditional ICON/MESSy integration compared to the current state of integration using ComIn. ICON dependencies are packed with the plugin to allow building the library and using an incremental development process.

6 Conclusions and Outlook

During the sprint, we created an initial version of a MESSy ComIn plugin to ease the connection of MESSy with ICON and to simplify the process of incorporating upstream changes from ICON. Although not complete for the whole model execution, the plugin comprises almost the full model initialization and state creation, serving as the basis for future expansion. We tackled several challenges, including handling dependencies on ICON, setting up a functional build infrastructure, and dealing with missing data and functionalities of ComIn. The requirements of the developed MESSy ComIn plugin also helped in shaping the Community Interface in terms of what should be supported and made available. We described the detailed process in a guide to serve as a potential reference point for other users interested in the use of ComIn. The guide is available on the natESM GitLab pages (see Loch et al., 2024).

We propose the following steps as the next course of action:

1. The remaining ICON dependencies in MESSy need to be resolved. This mainly requires adding MPI communication (patterns) to MESSy, which will be based on decomposition information provided by ComIn. The final goal is that the currently used unorthodox library approach is no longer required and the shared library MESSy/ComIn can be connected to ICON/ComIn following the ComIn approach.
2. A new build system based on CMake was introduced in MESSy, and the current build configuration, based on autotools, must be correctly ported and tested in the new build system.
3. A big next step is the addition of the MESSy time-loop subroutines to the MESSy ComIn plugin, which will allow the full model execution. A set of roadblocks have already been identified for this work, and at the time of writing this report, a sprint application for its continuation has already been accepted.
4. The developments were tested in technical terms during the sprint. To validate that the state matches the original ICON/MESSy integration, a robust scientific validation should also be performed with the current version of the plugin.

7 References

A full documentation can be found on the natESM Gitlab project:
<<LINK: should be open access for everyone>>

Jöckel, P., Sander, R., Kerkweg, A., Tost, H., & Lelieveld, J.: Technical Note: The Modular Earth Submodel System (MESSy) - a new approach towards Earth System Modeling, Atmospheric Chemistry and Physics, 5, 433–444, doi: 10.5194/acp-5-433-2005, URL http://www.atmos-chem-phys.net/5/433/2005/, 2005.

Jöckel, P., Kerkweg, A., Pozzer, A., Sander, R., Tost, H., Riede, H., Baumgaertner, A., Gromov, S., & Kern, B.: Development cycle 2 of the Modular Earth Submodel System (MESSy2), Geoscientific Model Development, 3, 717–752, doi: 10.5194/gmd-3-717-2010, URL http://www.geosci-model-dev.net/3/717/2010/, 2010.

Jöckel, P., Tost, H., Pozzer, A., Kunze, M., Kirner, O., Brenninkmeijer, C. A. M., Brinkop, S., Cai, D. S., Dyroff, C., Eckstein, J., Frank, F., Garny, H., Gottschaldt, K.-D., Graf, P., Grewe, V., Kerkweg, A., Kern, B., Matthes, S., Mertens, M., Meul, S., Neumaier, M., Nützel, M., Oberländer-Hayn, S., Ruhnke, R., Runde, T., Sander, R., Scharffe, D., & Zahn, A.: Earth System Chemistry integrated Modelling (ESCiMo) with the Modular Earth Submodel System (MESSy) version 2.51, Geoscientific Model Development, 9, 1153–1200, doi: 10.5194/gmd-9-1153-2016, URL http://www.geosci-model-dev.net/9/1153/2016/, 2016.

Kern, B. and Jöckel, P.: A diagnostic interface for the ICOsahedral Non-hydrostatic (ICON) modelling framework based on the Modular Earth Submodel System (MESSy v2.50), Geosci. Model Dev., 9, 3639–3654, https://doi.org/10.5194/gmd-9-3639-2016, 2016.

Loch, W., Hartung, K., Jöckel, P., Kerkweg, A. and Kern, B: MESSy/ComIn development guide, 2024. URL https://gitlab.dkrz.de/natESM/natesm_sprints_documentation/-/wikis/The-MESSy-ComIn-experience:-Considerations-in-creating-ComIn-plugins-out-of-complex-models