



Collaborative Climate Community Data and Processing Grid (C3-Grid)

DLR Workflow “Chemical Weather Forecast”

Arbeitspaket:	AP7 - Workflows
Autoren:	H. Bergmeyer, J. Hendricks, C. Kurz
Version:	Jul. 2008
Veröffentlichungsdatum:	Jul. 2008
AP-Koordination:	
Partner:	
Ansprechpartner:	H. Bergmeyer
Email:	Henning.Bergmeyer@dlr.de

Contents

1. Introduction.....	4
1.1. C3-Grid Model of Workflows.....	4
1.2. Objectives of this document.....	5
1.3. Overview.....	5
2. The Use Case “Measurements Campaign Support”.....	5
2.1. Objectives.....	6
2.2. Required data.....	7
2.2.1. Weather prediction data.....	7
2.2.2. Limitation of the ECMWF data base.....	7
2.2.3. Further data requirements	8
2.3. Resources for the Analysis Workflow.....	8
2.4. Visualization of the Model Results.....	9
2.4.1. Visualization Steps.....	9
2.4.2. Resources for Visualization.....	9
2.5. “Measurement Campaign Support” as a Workflow.....	9
3. The Visualization Workflow in Generation 0.....	11
3.1. Components.....	11
3.1.1. extract.sh (G0): An External Staging Script.....	12
3.1.2. plot_netcdf.sh (G0): Plotting with GMT.....	14
3.2. Limitations.....	15
4. Running the Model: The Analysis Workflow.....	17
4.1. Components.....	17
4.1.1. ECMWF data processing.....	17
4.1.2. Model Execution of ECHAM/MESSy.....	17
4.1.3. Post-Processing with CDO.....	20
4.2. Metadata Generation.....	21
4.2.1. Metadata Template for Campaign Products.....	21
4.2.2. Update of Metadata.....	21
4.3. Production Process during Campaigns.....	22
5. Accessing Model Results: The Visualization Workflow.....	24
5.1. Data Provider: Data Extraction from NetCDF Files with CDO.....	24
5.1.1. Requirements.....	25
5.1.2. Configuring Tools and Environment.....	26
5.1.3. Interpreting the Stage Request.....	26
5.1.4. Data Retrieval and Processing.....	27
5.1.5. Updating Metadata.....	29
5.1.6. Delivering data and metadata.....	31
5.1.7. Estimating Staging Time and File Size.....	31
5.2. Visual Plot with GMT and Ghostscript.....	33
5.2.1. Features and Requirements.....	33
5.2.2. Plot and Layout of NetCDF Data.....	35
5.2.3. Metadata Update.....	37
5.2.4. Extensibility.....	38
6. Conclusion.....	39
7. Appendix.....	43
7.1. NetCDF Stager (G1, based on PyModEst).....	43
7.1.1. Stager Starting Script: ipa_c3stager_mod.py.....	43
7.1.2. Data Processor for CWF-NetCDF Files: netcdf_extraction.py.....	43
7.1.3. NetCDF Tool Methods Module: netcdf_tools.py.....	49
7.2. plot_netcdf.sh (G1).....	51
7.3. Metadata for CWF Data.....	57
7.3.1. Metadata Template.....	57
7.3.2. Metadata Update XSL for CWF Result Data.....	75

1. Introduction

We describe here the C3-Grid oriented implementation of the components of the measurement campaign accompanying workflow of the Institute for Physics of the Atmosphere of the German Aerospace Center based on a *Chemical Weather Forecast* (CWF) model (ECHAM5/MESSy). We focus on inter component mechanics, system requirements and interfaces to the C3-Grid infrastructure as basis for the practical integration of the workflow into the C3-Grid. Before we describe the structure of the main part of this document, at this point we give a brief introduction to what the C3-Grid project understands as “workflows” and how they are supposed to be realized.

1.1. C3-Grid Model of Workflows

The C3-Grid’s centers of attention are sharing large-scale, geo-referenced, annotated data and tools to process and analyse it on a Grid infrastructure. These two main functionalities are made available to the virtual organization of the climate community on a web portal¹. While the sharing of data is accomplished by annotation of data with ISO metadata², centralized data management³ and making the data available through standardized data service interfaces⁴, the processing part is backed by distributed computing systems which provide software tools and computing power. These tools can be everything from small and simple scripts, over generic standard tools of the climate community (e.g. “Climate Data Operators”, CDO) to very specific and complex diagnosis and simulation codes.

One goal of the C3-Grid project is to enable the flexible use of standard tools on available data on the portal by providing them as an extensible tool set that users can work with on a step-by-step basis. The other goal is to allow descriptions of dependent execution steps of tools and possibly more specialized software like computational models as reusable schedulable process chains or trees, which are in general called *workflows*. The *Workflow Specification Language* (WSL)⁵ is the XML dialect that is used to describe workflows for the C3-Grid scheduler which executes them.

To provide a collection of software as tool sets, as well as to enable portal-based assemblage of workflows out of them, some kind of annotation of tools with metadata

¹ [B. Bräuer, “Portal”, Alfred Wegener Institute, Bremerhaven, Germany, Oct. 2006](#)

² [S. Kindermann, “C3Grid ISO 19115 Metadata Profile”, Deutsches Klimarechenzentrum, Hamburg, Germany, Feb. 2007](#)

³ [T. Langhammer and F. Schintke, “Datenmanagement-Dienst”, Konrad Zuse Institute Berlin, Berlin, Germany, March 2006](#)

⁴ [T. Langhammer and F. Schintke, “Archiv-Schnittstelle \(D\)”, Konrad Zuse Institute Berlin, Berlin, Germany, August 2006](#)

⁵ [Ch. Grimme and A. Papaspyrou, “Workflow Specification Language”, University of Dortmund, Dortmund, Germany, June 2006](#)

for identification, type, purpose, system requirements, resources they are provided by, input and output interfaces and other kinds of information to determine their function and availability is needed. An annotated software tool which is accompanied with capabilities to interpret and create C3-Grid formatted ISO metadata for its products can be called a *module*.

1.2.Objectives of this document

The on-going development of the grid application described in this document primarily aims at the realization of a C3-Grid workflow, some components of which may be suitable for later extension to modules for generic reuse in a toolkit. For the design of semantically correct annotations for components, a sophisticated understanding of their purpose and behavior is essential. Because of that in this document they are described regarding to the role in the workflow, recognizable dependencies from a C3-Grid point-of-view and towards module properties.

WSL workflow descriptions are part of the final stage of the project phase and will be documented in the final report. Concrete module annotations are developed as soon as components have been identified to fulfill common interests of the community and a module metadata format has been agreed on.

1.3.Overview

The second chapter describes the scientific and technical background of this workflow that consists of actually two loosely coupled sub-workflows, its purpose and how it is supposed to be used. To motivate the current generation 1 implementation of the workflow and embed it into the evolution of the C3-Grid, we briefly describe what has been learnt from a generation 0 prototype implementation of the visualization part in chapter 3. The next two chapters portray the current implementation of the first part, the “Chemical Weather Forecast” or “Analysis Workflow” and the second part, the “Visualization Workflow”. The last chapter draws a conclusion on what has been reached so far and what are the very next steps to address.

For the readers who like to have the source codes at hand for reference during reading, the most important ones are supplied in the appendix.

2.The Use Case “Measurements Campaign Support”

This chapter describes the purpose and supposed use of the application “Measurements Campaign Support” and the software components and resources, which it involves. This is the foundation and serves as guidance for the development of C3-Grid workflow components in the rest of this document.

2.1.Objectives

The workflow is supposed to be the prototype of an automated tool to support airborne field measurement campaigns. By combination of weather predictions, a global circulation model and an atmospheric chemistry model the user shall be supplied with daily predictions of the atmospheric composition ('chemical weather forecasts').

The user applies the workflow particularly for planning flight routes. Each hour of research aircraft operations implies very high financial expenses. Therefore the principal measurement region has to be chosen in advance, rather than during the flight. Furthermore many restrictions limit the free moving space of the aircraft, especially in foreign countries. Hence the approximate flight route has to be fixed days in advance in most cases.

The workflow has been developed as a platform independent application. During field campaigns which can last several weeks the daily availability of the predictions has to be guaranteed. The possible breakdown of individual computer resources has to be compensated. Furthermore, the workflow consists of several modular work steps of different complexity. Some of these work steps can be executed on small work stations, other require high performance super computing. The user in the field, who mostly has access to low capacity data lines, has no need to control the technical details of the workflow.

The workflow logically consists of two steps:

1. An automated routinely executed production job ('analysis workflow'). This part of the workflow is called daily to produce the current global chemical forecasts. It requires no interactions with the user.
2. A visualization workflow which allows the user to extract and visualize specific data (region of interest, altitude range, date and time of forecast, meteorological parameters, chemical species).

The model applied here is the ECHAM5⁶ global climate model developed by the Max-Planck-Institute for Meteorology, Hamburg, Germany. In this project the ECHAM5 model is applied within the MESSy⁷ domain (Modular Earth Submodule System) developed by the Max-Planck-Institute for Chemistry (MPI-Chem), Mainz, Germany. The MESSy framework allows the coupling of atmospheric chemistry to ECHAM5 and many other extensions.

⁶ Roeckner et al., MPI-Report 349, Hamburg, Germany, 2003

⁷ Jöckel et al., Atmos. Chem. Phys., 5, 433–444, 2005

2.2.Required data

2.2.1.Weather prediction data

A crucial requirement for the analysis workflow is prediction data of the atmospheric flow ('weather predictions'). These predictions are used to drive the global circulation model to enable the simulation of the chemical trace constituents by taking into account the weather conditions of the measurement day (so called 'nudging' technique). Only predictions of dynamical parameters are applied. Other parameters such as cloud or precipitation properties are calculated by the parameterizations of the global circulation model. The following quantities are required in terms of weather predictions:

- divergence of the wind field (spectral)
- vorticity of the wind field (spectral)
- temperature (spectral)
- logarithm of surface pressure (spectral)
- geopotential height of orography (spectral)
- optional: specific humidity (grid point)

The general circulation model (ECHAM5/MESSy) applied here is a spectral model. That is, the dynamical model quantities are represented in spectral space as a series of spherical harmonic coefficients. Consequently the prediction data have to be provided in spectral representation. A horizontal resolution of T106⁸ is recommended. Alternative resolutions are possible. The current version of the workflow operates in T42 spectral resolution.

The ECHAM5 model domain is vertically divided in several layers, so called model levels. The vertical σ -p coordinate system is applied. Since the weather predictions are also the results of an atmospheric model, the predictions are also used on model levels. This facilitates the interpolation of the data on the ECHAM5 grid.

2.2.2.Limitation of the ECMWF data base

Since ECHAM (ECMWF-model, version Hamburg) was developed from the weather prediction model of the ECMWF (European Centre for Medium-Range Weather Forecasts), the use of ECMWF predictions seems to be reasonable. However, this is handicapped by authorization rules. Only authorized users are allowed to use the data and derived products. Hence the use of ECMWF prediction requires an authorization management to restrict the use of the predictions and the resulting chemical weather forecasts to authorized users within the C3-Grid project. The authorization management was not yet implemented. Therefore, ECMWF data from previous years is applied for workflow development, rather than current weather predictions.

⁸ "horizontal resolution T106": triangular truncation at zonal wave number 106

2.2.3. Further data requirements

In addition to the weather prediction data, several data sets are required to drive the model:

- Initial and boundary conditions to drive the dynamical part of the model.
- Rate coefficients of heterogeneous chemical reactions.
- Concentrations of water vapour and several chemical trace constituents.
- Input data for photolysis rate calculation.
- Emission data for several chemical trace constituents.
- Input data for consideration of polar stratospheric clouds.
- Input data for radiative transfer calculations.

2.3. Resources for the Analysis Workflow

The analysis workflow is composed of two processing steps:

1. The Intera pre-processing which transforms the ECMWF data for use in ECHAM.
2. The ECHAM5/MESSy global chemistry forecast run.

Intera (Interpolating ECMWF Re-Analyses) developed by Ingo Kirchner (Institut für Meteorologie, FU-Berlin) enables the interpolation of any model analysis or prediction data on the ECHAM5 grid. The resulting data sets are used to drive the dynamics of the ECHAM5 simulations. For application of Intera the use of small servers is appropriate.

The following implementations have been realized within C3-Grid:

Server Name	Server Description	Server Info	Memory Usage	Cpu Time Usage
c3grid.pa.op.dlr.de	C3 grid server at DLR	Intel(R) Xeon(TM) CPU 3.20GHz, 2048 KB cache size	500 MB	700 CPU-s
ds7.dkrz.de	C3 execution host at DKRZ	IA-64, GenuineIntel Itanium 2, 1GHz	500 MB	5800 CPU-s

The chemistry-climate-model system ECHAM5/MESSy requires high performance computing resources. The following implementations have been realized within C3-Grid:

Server Name	Server Description	Memory Usage	Cpu's Used	Cpu Time Usage (per model day)
Hurrikan.dkrz.de	NEC SX6, DKRZ	6200MB	4	12500 CPU-s
a01.hlr2.lrz-muenchen.de	SGI Altix 4700, LRZ München, D-GRID Ressource	8500MB	32	720 CPU-s
cl.pik-potsdam.de	IBM p655 Cluster, PIK Potsdam	?	8	? CPU-s

2.4. Visualization of the Model Results

2.4.1. Visualization Steps

To visualize the model output the following steps have to be taken:

1. Projection of some variables of the result data that are still in spectral space to Gaussian grid.
2. Selection of geographic regions, pressure levels, species and points in time to be plotted.
3. Plotting the selection with suitable tools.

The ECHAM5/MESSy model produces output files in the NetCDF format. So for the first two steps the user can work with *Climate Data Operators* (CDO). The *Generic Mapping Tools* (GMT) are a command line toolkit that is suitable to plot geo-referenced NetCDF files as postscript files. In combination with *GhostScript*, step 3 can be fulfilled with a broad range of possible graphical output files for different purposes like showing in web browsers or for printing.

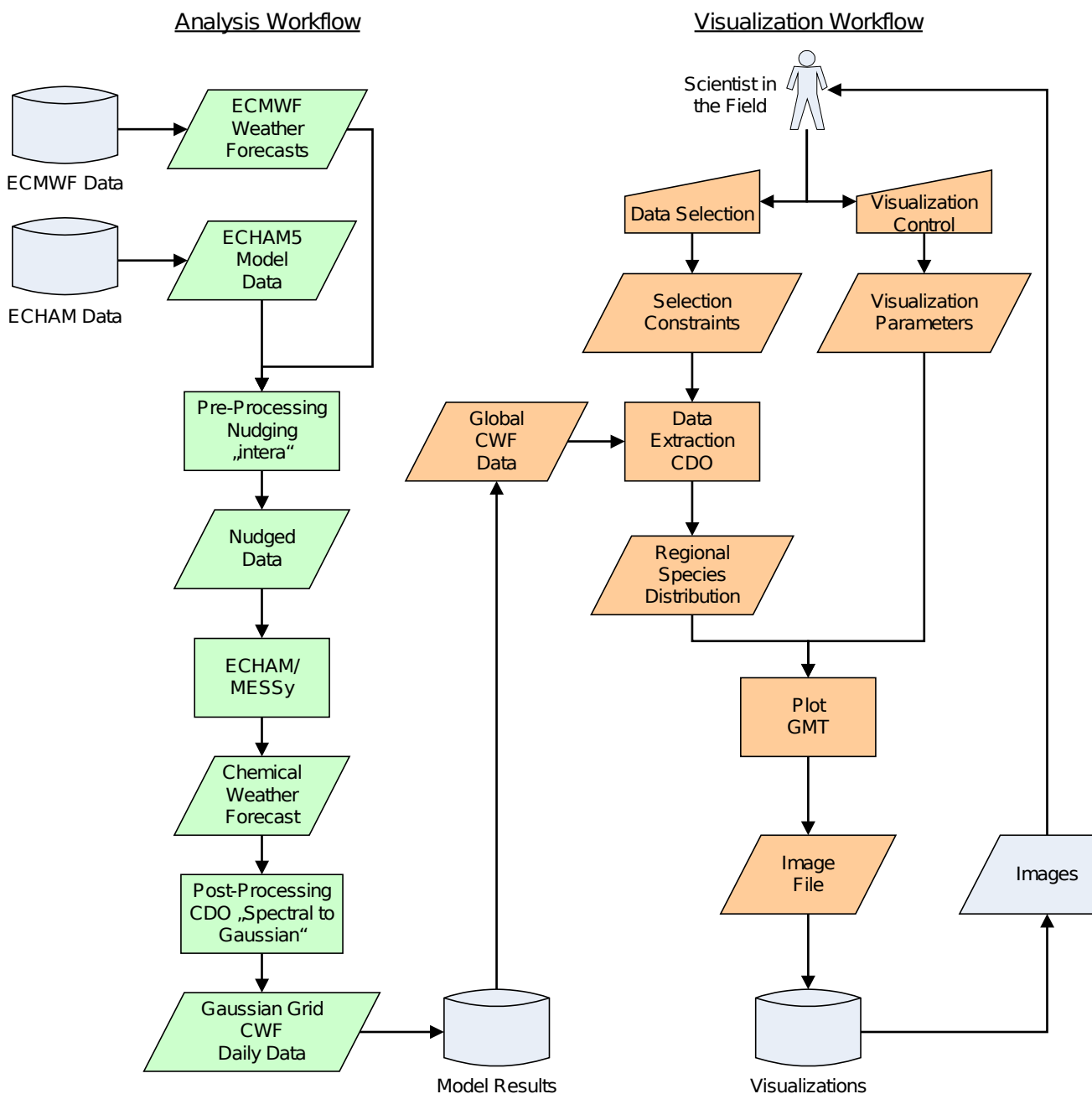
2.4.2. Resources for Visualization

In general the scientist in the field would remotely connect to the server containing the forecast results and perform the visualization steps by hand before downloading the image or even reduced data to a local PC or to a laptop computer. The data selection and plotting processes are not as performance hungry as the model run and usually can be executed on PCs in acceptable ranges of time, but the amount of data to be transferred still may be a challenge for low-data range internet connections in the field. In addition the user needs to have fundamental knowledge about proper usage of the tools.

2.5. “Measurement Campaign Support” as a Workflow

The identified active components of the described application are dependent on each other regarding to their respective input and output files. shows these dependencies as a flow chart, identifying as well the two sub-workflows.

The green-colored components of the analysis workflow are run once a day during the campaign period and produce forecasts for the following days. The orange components form the visualization workflow, which is asynchronously performed by a scientist partly remotely and on locally available equipment. The two workflows are loosely coupled by a storage server that persists model results for world-wide availability.



The C3-Grid provides means to realize the workflow, making most of its components transparent to the user while keeping the relevant features accessible to the user. The components of both sub-workflows are implemented to be executable on D-Grid compute resources, while only the visualization part need to have a graphical user interface (GUI) in shape of portlets on the C3-Grid portal.

Because the chemical weather forecasts as short term forecasts have only a limited time validity and this data is only planned to be used in this application context for visualization, the visualization step 1 mentioned in section 2.4.1 for conversion of data from spectral space to gridpoint data can safely be realized as a post-processing step after the model execution, saving processing time at visualization time.

Figure 1: Measurement Campaign Support Fow Chart

3. The Visualization Workflow in Generation 0

During the prototypical phase of the C3-Grid project, which is internally associated with the term “Generation 0”, a first version of the visualization part of the measurement campaign workflow had been implemented to prove its realizability, acquire knowledge about implementing workflows and data providers in the C3-Grid, determine requirements and get general user feedback. It ran successfully in the portal, creating nicely laid out color plots of pre-generated model result data and was demonstrated on a user workshop in June 2007 in the “MARUM”, Bremen.

The demo allows the specification of a selection from an example model result data set and to request its visualization. Usually after less than a minute the picture is automatically displayed on the portal.

3.1. Components

At this point we only describe the basic concepts of the prototype implementation. It mainly consists of two bash scripts, “extract.sh” which implements the functionality of a C3-Grid data provider interface and “plot_netcdf.sh” realizing the plot functionality. The prototype implementation used the C3-Grid data provider at Zuse Institut Berlin (mardschana.zib.de) and the compute provider of the Potsdam Institute for Climate Impact Research (c3-grid.pik-potsdam.de), each backed with the scripts explained in the following.

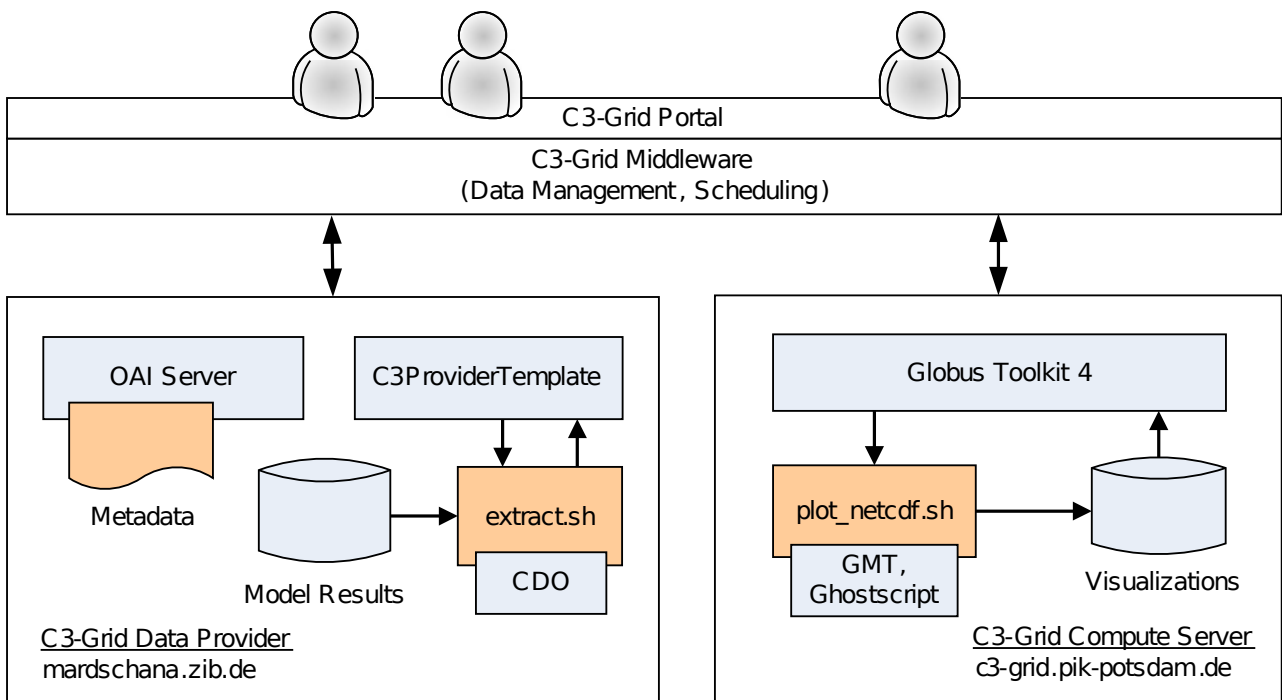


Figure 2: Visualization Workflow Specific Implementations (orange)

The data is made available on the portal like any other data on the C3-Grid, through

installing a data provider web service interface and publishing the according metadata on an OAI server for harvest by the portal. For instructions on how to install and configure the web service *C3ProviderTemplate* and the *OAI server*, refer to the C3-Grid document “How to setup a C3-Grid Data Provider Site”⁹. The data and data retrieval specific features of the web service are usually implemented as additional scripts, so-called “External Stagers”, as it is “extract.sh”.

The installation of the visualization script works like any script installation on a linux account, just that it and all the tools it needs have to be executable by the C3-Grid user on the compute resource. Any required environment settings have to be provided as *environment modules*, which in turn have to be mentioned in the workflow description.

3.1.1.extract.sh (G0): An External Staging Script

The attribute “external” refers to the fact, that the data provider web service implementation in Java “C3ProviderTemplate” can be configured to launch independent scripts to realize the local adaption of C3-Grid staging functionality of data access, trimming, merging and transferring using the tools, which are available probably only at the specific data provider’s site.

The role of external staging scripts is to interpret the constraints received in a C3-Grid stage request, transform them to some form of requests to a data storage system (databases and file systems), package and annotate the results with ISO metadata and transfer them to a stage request specific target address. In addition stage requests may be tagged as estimation requests (“JustAsk=True”), with which the data provider is asked for worst-case estimations of the required time to process and deliver the data, as well as a prediction of the size of the file to be produced.

Even though the strategy to implement this functionality is arguable by the data provider and depends heavily on the underlying storage system type and interfaces, there is a procedure which seems to be suitable for many of such use cases.

1. Decide upon the list of *object identifiers* which storage locations will have to be accessed and in what way.
2. Choose a time and temporary storage space efficient retrieval strategy for the base data on format and granularity properties of the data which is retrieved from the storage systems. This is data provider specific information that the staging script needs to “know”. It is advisable to keep the staging system modular in terms of retrieval and processing strategies.

⁹ [S. Petri, “How to setup a C3-Grid Data Provider Site”, Potsdam Intitute for Climate Impact Research, Potsdam, Germany, Feb. 2007](#)

3. Merge the data into one result file.
4. Produce a local copy of the metadata which describes the used base data
5. Modify the local metadata copy to reflect the properties of the result data and distinguish it from other results (identification). The metadata contains as well lineage information about the production process of the data and the origin of the data that was used to produce it.
6. Transmit data and metadata to the target destination with GridFTP.

The C3ProviderTemplate can deliver the stage request parameters it receives over the SOAP interface in several formats: As XML structure, in Java property file format and as Ini file format with simpler property names. They are either directly piped to the script on the standard input channel or written to a parameter file which can be randomly accessed.

The extract.sh script takes an ini-file as input via the command line parameter “-i <path>” and optionally allows the specification of a time log file via the parameter “-t <path>” for output of runtime logging. The latter serves processing time measurement of the different parts of the scripting as foundation for runtime estimations, which is a required feature for full data provider functionality compliance.

Our generation 0 bash script implements the before mentioned procedure for only one filesystem as data source and only the CWF dataset in NetCDF. The data is sliced along the timeaxis into files containing eight timesteps of geographically global data of several chemical atmospheric tracers, which are accessed on the file system via temporary symbolic links in the scratch area and processed one by one. The metadata delivered with the file is only a copy of the parameter ini file, because “sourcing” it in the plot script as information about the data is easier than reading the needed properties from the data file there. This prevents full compliance of the prototype to the C3-Grid data provider specification, but serves its sole purpose to rapidly develop the workflow demonstrator.

The most important software tools for manipulation of NetCDF files of the CWF result data are the *Climate Data Operators* (CDO), which to have installed and configured with *Module Environment* settings, are an agreed basic installation requirement for data providers. The extract script relies on CDO, a fully configured C3-Grid environment and the Bash shell to exploit it.

Requirements in short:

- CDO installed with NetCDF support
- C3-Grid environment

- o directories of the workspace
- o environment variables (provided by Module Environment)
- o standard tools (Globus, Module Environment)
- o base data files stored it `$C3_WN_DATA/dlr`
- o scratch directory `$C3_WN_SCRATCH/dlr_campaign`
- o existing target directories for results

The development of the bash script has been abandoned to follow a modular approach in Python based on the things learnt from generation 0, directly supporting all of the C3-Grid features and requirements.

3.1.2.plot_netcdf.sh (G0): Plotting with GMT

The plot script was developed to take exactly one NetCDF file and a parameter ini file as inputs and create a *Portable Network Graphics* (PNG) file of it for immediate display on the C3-Grid portal. Resource estimation, as for data providers, and SCP transfer for the results had been implemented but were actually never used.

In addition to the requirements of the extraction script, the following software is needed:

- Generic Mapping Tools (GMT)
- Ghostscript

There are only few C3-Grid specific constraints that the implementation of the plot script has to fulfill to be C3-Grid compliant, such as metadata production. In addition to that only image size and file format had been essential at the point of its development to let it be run on the C3-Grid and to display the picture afterwards properly in the portal. There was no ISO metadata created for the picture, because the visualization workflow finishes quickly and the picture is not persisted or otherwise published independently of the individual visualization workflow run.

The worth of the script rests in its automatic layout method, which fits plots of extracted regions of any aspect ratio onto a virtual DIN-A4-sized sheet and arranges the title, palette, annotations and legend accordingly around it. The PNG file is rendered from a temporary postscript file at a resolution of 96 dpi and trimmed to show only a slim frame of empty background around the contents. This leads to pictures which are always less than 600 pixels wide and generally not more than a 830 pixels high.

The used tools offer a lot of flexibility according to output possibilities. It is easy to adjust the script for output of different file formats, such postscript for high quality vector based printing. Because of this straight behavior and independency of the

implementation from the C3-Grid environment, a lot of “plot_netcdf.sh” can be transferred to generation 1. The only fundamental change needed is a decoupling of the parameterisation from the extraction script.

More details about this change, the functionality and compatibility with generation 1 are described in section 5.2. The source code of the current version of the script, that is used in generation 1 is available in the appendix 7.2.

3.2.Limitations

At the time of generation 0 jobs on the C3-Grid were still managed by a very static scheduling system prototype and workflow specifications were not yet in use. While XML ISO metadata was already used to make data searchable in the portal, to download selections or start workflows on it, workflow generated metadata was not yet used on the Grid.

So the workflow components for the CWF workflow were simply realized as specific shell scripts around the tools *CDO* and *GMT*. They were designed to be extensible, but only so far as to help getting the tools working on the grid and finding the optimal combinations of parameters to create useful images. Metadata processing was never implemented for that version, because development towards a generic metadata toolkit was already on the way at partner institutes. We focused on a close coupling of the data provider and the plot script by transporting “shell sourceable” parameters from the data extraction script to the plot script on a different server. Manual generation and interpretation of XML documents in shell scripts had been done by other project partners already and was considered a dead branch of research for the development of this workflow, compared to the expected availability of a proper metadata toolkit, that was going to make the complete XML processing transparent to the implementer of the data service and modules.

Things learned from that generation are requirements and implementation suggestions for a generic execution environment that eases the adaption of existing processing tools for use on the C3-Grid and decouples the executable components from another. There are recurring tasks like

- loading, interpreting and modifying metadata (requires a generic toolkit),
- translating CF variable names to data table identifiers and indices and back again (requires a translator or thesaurus component, at best based on a centralized database),
- interpreting, executing and replying to stage requests,

- interpreting configuration files and environment variables to manage temporary workspace and files,
- transferring data and accessing databases,
- error logging and usage statistics (should be done centralized),
- estimation of result file sizes and processing time (especially time estimation is far more difficult than expected)
- tool and task specific calculations and adaptations. (e.g. for image layout)

The first suggestion derived from this is to use a more sophisticated scripting language which is easier to handle, extend and test than shell scripts, whenever recurring C3-Grid system- or file-oriented operations have to be implemented. We recommend Python as it is available for most C-enabled platforms, easy to learn and use, can do mathematical computations and offers a broad range of features to access and manipulate lists, maps and strings, as well as community supported open source tools for processing of XML and specific scientific data formats, such as NetCDF and HDF.

The second suggestion is the development of a generic library that supports data and compute providers connecting their tools and data to the C3-Grid.

This work has begun and has taken shape as a modularized Python library for external stagers (*PyModEST*), which has at first been implemented for the WDC-RSAT C3-Grid data provider of the German Aerospace Center and is now used for the Chemical Weather Forecast data provider, as well, where it has significantly reduced complexity and increased readability compared to the former bash script version. More in-depth information about the library can be found in another C3-Grid deliverable document “The WDC-RSAT Data Provider – A Modular External Stager in Python (PyModEST)”¹⁰

¹⁰ B. Hildenbrand and H. Bergmeyer, “The WDC-RSAT Data Provider – A Modular External Stager in Python (PyModEST)”, German Aerospace Center, Cologne, Germany, July 2008

4. Running the Model: The Analysis Workflow

4.1. Components

4.1.1. ECMWF data processing

The ECMWF raw data has to be supplied directly by the ECMWF. The required authorization management for using the data has to be negotiated with the ECMWF. The data download has to be realized following the ECMWF guidelines. In the current version of the workflow, data from previous years is applied which is appropriate for current workflow development purposes.

For use of the ECMWF data in the ECHAM5 simulations relevant meteorological parameters have to be extracted and interpolated on the ECHAM5 grid. The quantities to be extracted are:

- divergence of the wind field (spectral space)
- vorticity of the wind field (spectral space)
- temperature (spectral space)
- logarithm of surface pressure (spectral space)
- geopotential height of orography (spectral space)

To this end, the Intera script `preprocess_using_tasks.csh` was developed which requires the following input/changes for application:

- Base and working directories have to be specified (e.g., `set c3_Basedir=~/.c3; set Workdir=$c3_Basedir/workdir`).
- The date (year, month, day) of prediction days to be preprocessed must be specified. This information has to be provided within the script (`set CurrentDay=xx, set CurrentMonth=xx, set CurrentYear=xxxx`).
- The ECMWF raw data has to be supplied. The corresponding path has to be specified within the script (`set Incoming= ... path ...`). The data used here (data from previous years, rather than current predictions) is transferred by grid-ftp within the script. The corresponding ftp controls have to be adjusted or replaced where appropriate.
- Links to Intera have to be specified within the script (e.g., `$c3_Basedir/bin/intera`).
- The file `template.echam$HorizontalResolution$VerticalResolution` (e.g., `template.echamT42L41`) has to be linked to the Intera calls in the script by specifying the corresponding path and filename. The file contains all necessary information about the ECHAM5 model grid and its vertical structure.

The interpolated data is written in the current working directory.

4.1.2. Model Execution of ECHAM/MESSy

The model driver job “xforecast” which was adapted here for specific application in C3-Grid is called from the workflow script. To generate the model executable file called from xforecast the ECHAM5/MESSy (MPI-Chem, see section 2.1) modelling framework

has to be implemented on the respective computer resources. This has to be realized in accordance with the guidelines and licence agreement of MPI-Chem. The particular model configuration (e.g., spatial and temporal resolution, chemistry scheme, nudging) has to be implemented according to the requirements of the xforecast job.

The model is called separately for each forecast day. The restart of the model after each simulation day is organized by the workflow script workflow.csh. The particular forecast period has to be set in the xforecast script (e.g., START_YEAR=2006, START_MONTH=05, START_DAY=17; STOP_YEAR=2006, STOP_MONTH=05, STOP_DAY=19). Additionally a specific directory structure as well as a set of input files is necessary for the execution of xforecast:

Directories

The workflow infra structure has to be implemented in the base directory:

```
$HOME/c3
```

The “workflow.csh” script as well as the model job “xforecast” are operating from the directory:

```
$HOME/c3/workflow/
```

The following working directories have to be created:

```
$HOME/c3/workdir and $HOME/c3/workdir/run
```

Executables

The model has to be compiled separately before using the workflow. The executable file E5M1 has to be placed in the directory:

```
$HOME/c3/workflow/bin/
```

Namelist

A set of namelist files is necessary to operate the model. Templates of these namelists are part of the MESSy modelling framework. The namelists were adapted for operating xforecast within C3-Grid and are stored in the directory:

```
$HOME/c3/workflow/nml
```

Input data

Fixed input data for calculation of atmospheric dynamics:

The operation of the model requires several fixed input data sets (initial and boundary conditions) for the simulation of atmospheric dynamics. These data include parameters such as sea surface temperatures, sea ice coverage, concentrations of greenhouse gases, and many others. These data have to be chosen in accordance to the respective grid resolution/structure of the model. The data (NetCDF) have to be stored within the directory:

```
$ESH_INIR00T/echam5_ini.
```

ESH_INIR00T, its subdirectories, as well as the specific file names have to be specified within the xforecast script.

Fixed input data for calculation of atmospheric chemistry and radiation:

The model further requires fixed input data for the calculation of atmospheric chemistry and radiation. A number of input files are necessary containing the following information:

- Concentrations of several chemical compounds
- Rate coefficients and related parameters for heterogeneous chemical reactions
- Input data for calculation of photolysis rates
- Emission rates
- Data for consideration of polar stratospheric clouds (PSC)
- Input data for radiation calculation

These data (NetCDF) have to be stored in a directory \$ESH_INIR00T/messy (ESH_INIR00T has to be specified in "xforecast"). The location and names of the respective data files in this directory have to be specified within the namelist files of the respective submodule of the model. Some additional input data for the radiation calculations has to be stored in the directory \$ESH_INIR00T/echam5_ini. For more details, see the current documentation of the MESSy system.

Weather prediction data:

The weather prediction data necessary for applying the nudging technique (see section 2) has to be stored within the directory:

```
INPUTDIR_NUDGE=$HOME/c3/workdir/nudge
```

For each forecast day the following files generated as described above are required:

```
${FNAME_NUDGE}_vor (vorticity of the wind field)
```

```
${FNAME_NUDGE}_div (divergence of the wind field)
```

```
${FNAME_NUDGE}_stp (temperature and logarithm of surface pressure)
```

```
${FNAME_NUDGE}_sst (sea surface temperature)
```

FNAME_NUDGE is composed of date, horizontal, and vertical resolution (e.g., FNAME_NUDGE =20060517_T42L41).

Output data

The output is written in the working directory:

```
$HOME/c3/workdir/run
```

Each model run produces a large set of output files in NetCDF-format, each containing data related to specific physical or chemical processes represented by the model. The output of chemical species concentrations is written in the file:

```
forecast__yyyymm.dd_tracer_gp.nc
```

where yyyy, mm, and dd denote the year, month, and day of the forecast. Many other files are written containing, for instance, output of dynamical parameters, cloud properties, or radiation parameters. These data however are currently not subject of the workflow but could be included in future versions.

The chemistry output file `forecast__yyyymm.dd_tracer_gp.nc` contains the following information:

- Three-hourly resolved global three-dimensional (horizontal, vertical) distributions of chemical constituent concentrations. The constituents covered by the data set are: N₂O, PAN (Peroxyacetylnitrate), N₂O₅, CO, HNO₃, CH₄, ISOP (Isoprene), ClONO₂, NO, HO₂, HCl, O₃, NO₂, and OH.
- Information to transform the concentrations from model levels to pressure levels: surface pressure and logarithm of surface pressure, several coefficients for coordinate transformation. The transformation can be achieved using the CDO `ml2pl` command.

More information about the chemical output data can be gained from the NetCDF file.

4.1.3. Post-Processing with CDO

In C3-Grid geographic regions are by standard denoted as bounding coordinates on Gaussian grids (longitude, latitude). The chemical forecast data resulting from the model simulations is provided on such a grid. Hence no grid transformation operations are necessary for the chemical constituent concentrations. However some dynamical quantities, particularly the log surface pressure are provided in spectral space. To support fast processing at the data provider these spectral data have to be converted to Gaussian grids.

```
cdo sp2gp $ModelResultFile $OUTFILE
```

4.2. Metadata Generation

This workflow serves to produce data once a day during a campaign period, updating a data provider with current forecasts of chemical tracer distributions for the following few days. While the workflow neither retrieves any data from C3-Grid data providers, nor its components use any ISO metadata during their execution, the result of the workflow has to be annotated with ISO standard metadata for use on the C3-Grid. As soon as the metadata is checked into an OAI server that is registered at the C3-Grid data management system, the data becomes searchable on the portal, which regularly harvests the metadata from the OAI servers.

4.2.1. Metadata Template for Campaign Products

The fact that all data products originate from the same sources, result from the same production process and contain the same attributes leads to the idea to create one metadata file template for the whole campaign, that is used to create updated copies associated with new results by a unique identifier and with what really differentiates them from products of the days before.

The template for the chemical weather forecast data follows the “C3Grid ISO 19115 Metadata Profile” and appendices. The full XML document can be found in the appendix 7.3.1.

4.2.2. Update of Metadata

During one measurement campaign the workflow is executed once a day and produces results, which differ from the results of the days before in the up-to-date weather forecast input data for the nudging process, the calculated tracer distribution results and the respective time period. The input data is described in the Lineage section of the metadata template and the set of contained variables is constant.

The only differences that have to be recorded in the metadata are the dates, the results belong to and the object identifier of the data set.

The start and stop dates can be retrieved as first and last element of the date list “cdo showdate” produces from the result data as shown in Figure 3. The dates CDO replies are formatted like “yyyy-mm-dd, hh:mm”, so to comply with ISO8601 they have to be formatted to “yyyy-mm-ddThh:mm:ssZ”

```
DATES=$(cdo showdate $INFILE)
START_DATE=${DATES[0]}
ri=${#DATES[*]}
ri=$((ri-1))
STOP_DATE=${DATES[$ri]}
START_DATE=${START_DATE%,*}T${START_DATE##*,}:00Z
STOP_DATE=${STOP_DATE%.*}T${STOP_DATE##*.}:59Z
```

Figure 3: Retrieving the Time Period from NetCDF Files

The metadata file then is created by an XSL transformation through invocation of “xmlstarlet” (Figure 4).

```
xmlstarlet tr $MD_Update_XSL
-s size=237322188
-s start=$START_DATE
-s stop=$STOP_DATE
-s version=$VERSION_TEXT
-s constraints=$BASEDATE $FORECASTLENGTH
$MD Template > $MD BASENAME $BASEDATE $FORECASTLENGTH.xml
```

Figure 4: Metadata Update for CWF Data

The XSL script updates the time period values and adds a lineage entry based on the “version” and “constraints” properties. Because every day a forecast is done for several days, \$BASEDATE and \$FORECASTLENGTH are used to define, from what day the underlying basedata originates and how many days towards the future the current forecast result is dated in relation to \$BASEDATE. This helps to group results by reliability, because this factor decreases with increasing forecast length.

4.3. Production Process during Campaigns

In contrast to modules and diagnostic workflows, the analysis workflow is not meant to be executed manually by a portal user. It has to be started automatically once a day during the campaign period. This can comfortably be accomplished by a cron job that calls a workflow script, which sets simulation parameters according to the current day and then executes the process chain of pre-processing, the model and the post-processing consecutively.

For the productive system an additional step at the end of the workflow script realizes the coupling of analysis workflow and visualization workflow. The data is published on the C3-Grid by placing the metadata on an OAI server and making the results available through a data provider in the C3-Grid. This step is currently in development in the C3-Grid project.

Another step to integrate the analysis workflow with the C3-Grid is its execution over the scheduling system. For that the workflow script be transformed into an abstract workflow description in the C3-Grid WSL. The WSL description still will have to be adapted every day for each current date before submission to the scheduler by a cron job, but this method allows a more flexible use of distributed Grid resources and is the designated way to the extension of C3-Grid data management for automatic, secure data publishing and ingestion processes.

With the described components the analysis workflow presents itself as a good

example for an automatic data production system for the C3-Grid.

5. Accessing Model Results: The Visualization Workflow

The Visualization Workflow serves the purpose of making the results of model runs available to the users of the C3-Grid portal. The intended use for flight route planning requires means to generate graphical plots of the predicted distribution of chemical tracers in selectable campaign regions.

For this a data provider service has been configured to deliver the result data, and a workflow module is developed that produces plots of this data in various file formats and document sizes. An abstract workflow description is realized with parameters for region, tracer variables, color palettes and so on, which are set by the portal user to generate a concrete workflow description instance for interpretation by the C3-Grid scheduler to issue data requests to the service, calls to the plot module and determine the necessary data transfers. During the processing, metadata is updated to reflect the transformations of the data. In this way the visualization workflow is a world-widely available web portal application running on the grid, which can be used by the scientist in the field, as long as there is an internet access available.

The following two sections describe the wrapper scripts around the used tools as well as their installation and configuration, beginning with the functionality of the data provider, where the data about requested chemicals contained in selected geographic regions is extracted from the result data. The second one is the plot script, that creates nicely laid out picture files from the extracts for web display and high quality printing. The third section deals with the schedulable workflow description and its parameterization through the portal.

5.1. Data Provider: Data Extraction from NetCDF Files with CDO

This section describes the configuration of the C3-Grid data provider service and the implementation of the extraction functionality for the result data of the ECHAM/MESSy model.

The service is realized with the common C3-Grid Data Provider Template V0.8 that is configured to launch the estimation, extraction and staging functionality for this data as an external staging script. This staging script is implemented in Python re-using the modularized external stager library PyModESt that was originally developed for the WDC-RSAT data provider of the German Aerospace Center. The NetCDF transformations are carried out using CDO.

5.1.1. Requirements

The post-processing step of the Analysis Workflow described in the previous chapter produces NetCDF files describing the predicted global distribution of several chemical tracers in the atmosphere. For this data to be fed into the C3-Grid for general availability, the non-optional feature set specified for data providers has to be implemented:

- Selection of chemical tracers to be delivered, referred to by CF names, if possible
- Extraction of a geographical region with bounds along longitude and latitude axes
- Extraction of layers of the atmosphere, specified by lower and higher bounds in pressure values.
- Extraction of all data available for a requested time interval
- Generation of updated metadata for the delivered data extract
- Prediction of extract file size and processing time

The structure of the model result data leads to several implicit implementation requirements that have to be taken into account:

- File merging along the time axis: The data is not uniformly available in a data base, so the requested data cannot just be generated like a “view” with a data base transaction. It is located on a file system, sliced into *base data files* containing data of one predicted day as eight *time steps* of three hours length each. The extracted data is supposed to be contained in one file. Because the requested time interval may span across several time steps, even across several base data files, some resource saving way of merging parts of files into one has to be implemented.
- The variable names used to refer to the chemical tracers in the base data are no CF names like “*mole_fraction_of_ozone_in_air*”, but chemical acronyms like “*O3*”. The CF names and pseudo CF names used on the C3-Grid level (e.g. in data requests and ISO metadata files) have to be interpreted.
- Within the data files pressure levels have the unit *Pa* meanwhile in the C3-Grid *hPa* are communicated. This factor of 100 has to be applied when parameterizing CDO.

5.1.2. Configuring Tools and Environment

The NetCDF data files are processed with *CDO* and the staging script is written in *Python*. To manage the settings of the execution environment the *Modules* tool is used.

1. Install Modules
2. Install C3ProviderTemplate
3. Install NetCDF library
4. Install CDO
5. Install Python
6. Install Python API for modularized external stager (pyModEst)

5.1.3. Interpreting the Stage Request

The pyModEst API offers a set of modules that do the major work of interpreting received StageRequests and preparing the environment for the staging task accordingly. The jobs left to the developer of staging scripts are deciding, which data processing method is going to be used for which identified base data set and implementing the data processors.

```
from c3grid.c3dataprotider.c3stager import C3ExternalStager
import netcdf_extraction

PROCESSOR_TYPES = {"de.dlr.ipa.CWF" : netcdf_extraction.IPA_NCDF_Processor}

if __name__ == "__main__":
    C3ExternalStager(PROCESSOR_TYPES)
```

Figure 5: ipa_c3stager_mod.py

Figure 5 shows the contents of the main Python module that is executed by the provider template, whenever a stage request is received. The sole purpose of this code is to start a general external stager *C3ExternalStager* with a mapping of *file identifiers* to data processors. In this case there is only one map entry that maps the data set identified by “de.dlr.ipa.CWF” to the data processor *netcdf_extraction.IPA_NCDF_Processor*.

The *C3ExternalStager* retrieves the stage request in property file format from the standard input, compares the elements of the list of object identifiers delivered in it with the defined file identifiers in the map and calls the appropriate methods of the corresponding data processor, dependent on the kind of request, a concrete stage request or an estimation request. More about how the *C3ExternalStager* is realized is going to be available in another C3-Grid deliverable.

5.1.4. Data Retrieval and Processing

The data processor implements the data download from the persistent storage location, extraction of the requested data from each of the downloaded files and merging of that data into the result file. During this process, all changes that have to be reflected in the ISO metadata of the result, are noted as properties for later metadata update.

The C3ExternalStager automatically provides to the data processor an individual workspace for temporary files, the constraints from the stage request as properties and access to an initialized metadata object that can be updated with convenient methods. As long as the author of the data processor implementation uses the provided file paths to store the resulting data extract at, the upload of the data files is taken care of automatically, as well is garbage collection and detection of repeated requests. This is how it is done for the CWF data processor.

Initialization

As mentioned before, the software on the C3-Grid level communicates about data using ISO metadata and CF-names to identify variables. In the low-level context of data processing in most cases other metadata formats and variable naming standards are used. To determine, what variable name or table index refers to the requested data in the files, some means of translation between data context and C3-Grid context have to be provided.

For this purpose during initialization of the data processor a C3thesaurus instance is created, which provides methods for a simple two-way mapping between variable identifiers in files and C3-Grid variable names that follow the CF convention. Figure 6 shows how this mapping is currently set up.

```
C3_CWF_SCOPE_MAP =
  { "de.dlr.ipa.CWF" :
    { "mole_fraction_of_carbon_monoxide_in_air"      : "CO",
      "mole_fraction_of_carbon_dioxide_in_air"      : "CO2",
      "mole_fraction_of_hydroperoxy_radical_in_air" : "HO2",
      "mole_fraction_of_hydroxyl_radical_in_air"    : "OH",
      "mole_fraction_of_nitrogen_dioxide_in_air"    : "NO2",
      "mole_fraction_of_nitrogen_oxide_in_air"      : "NO",
      "mole_fraction_of_ozone_in_air"               : "O3",
      "mole_fraction_of_peroxy_acetyl_nitrate_in_air" : "PAN",
      "geopotential"                                :
    }
  "geosp",
    "surface_air_pressure"                          : "aps"
  }
}
```

Figure 6: Variable Name Mapping between C3-Grid and ECHAM Data Scopes

Now the list of requested variables can be translated by the thesaurus from C3-Grid

scope to the local scope. Should any variables not be translatable, warning messages will be logged. If the thesaurus was not able to translate any value, the stager will break down at this point with an exception.

Then local property copies of the relevant request constraints are created. Because of quantisation effects the extraction result can in most cases not precisely match the request constraints. The local values can be adjusted to the actual properties of the result and thus be reflected in the metadata for consistency.

In this stager implementation only the first object identifier from the allowed list in the stage request is going to be minded and thus copied for the creation of an updated object ID. Then follow copies of longitude, latitude und altitude bounds for the spacial extraction operation. The only accepted values for altitude constraints have the unit *hPa*. Any other requested units are responded to with an exception. The altitude bounds are normalized to *Pa* by multiplication with a factor of 100. Finally the copies of the time interval bounds are created.

Data Retrieval, Extraction and Merging

The base data files in this stager environment are accessible on the local file system, but for easy extensibility towards transfers over networks, the method `urllib.urlretrieve()` is used for downloads into the local workspace at the path `self.c3env.workdir + src_file_name`. The files are identified by a name pattern based on the date that the tracer data is associated with.

```
SOURCE_FILE_NAME_DATE_PATTERN = "forecast__%Y%m.%d_tracer_gp.nc"
```

For example: "forecast__200806.24_tracer_gp.nc".

Each file contains eight time steps that are accessed by an index number in the range $[0, 7]$. Thus the hour-minute-second part of the requested time bounds are quantized to bounds of three hour intervals. Because of the interval containment policy agreed within the C3-Grid project, that determines that boundary constraints of intervals are strictly to be interpreted as outer bounds, the decimals of the index calculation are rounded up for the beginning bound and rounded down for the ending bound.

To save storage space during processing, every base data file that covers the time interval is downloaded, extracted and merged with the result file individually. Any unavailable base data file produces an exception and leads to a controlled, garbage collected break-down of the stager and a warning message replied to the requesting party.

Execution of tools as sub-processes from Python requires initialization of the specific

environment each time. For this purpose the software “Environment Modules” is installed and configured. The call of CDO once for each downloaded file is done with a command line like this “module load c3grid cdo ; cdo ” appended with the following CDO parameters:

Variables: `-selvar, var1, var2, ...`

Time Steps: `-seltimestep, t1, t2, t3, ...`

Region: `-sellonlatbox, lon_min, lon_max, lat_min, lat_max`

Pressure Level: `-ml2pl, alt_min_pl, alt_max_pl`

The pressure level extraction is an interpolation from model levels to the specified pressure level. This operation requires the base data to contain a surface pressure variable (“lsp” or “asp”). The parameters “alt_min_pl” and “alt_max_pl” will be interpreted by CDO as individual pressure level requests. Model level data, that is contained in the base data and refers to space between the these two altitude bounds only contribute to the interpolation towards the specified bounds, but there are no additional pressure levels added to increase the special resolution in between. A request with two equal bounds would lead to the inclusion of two equal pressure level data sets into the extract, so this case has to be detected to interpolate only once.

For the first day this directly creates the result file at `self.c3enc.localbasefile`. From the iteration for the second day of the interval on, the data is first written into the temporary file at `self.c3enc.localtempfile` which is then merged with the result file using the CDO command:

Merge: `mergetime`

When the data has successfully been merged, it is possible to realize a file format conversion, in the case the data had been requested in another format than NetCDF. The requested format identifier is available in the string property `self.stage_request.targetfileformat`. Even though no conversion is currently implemented, this property will be checked and compared to the output file format NetCDF, to at least log a warning in case of a mismatch.

5.1.5.Updating Metadata

PyModEST provides some convenient methods to update metadata for extracted data. To accomplish this, the C3ExternalStager automatically downloads the corresponding metadata description of the base data that is selected by the object identifier in the stage request from the OAI server that has been specified in the C3ProviderTemplate configuration file “c3grid.provider.properties”.

By realizing the method “updateMetadata(self, md)” in the data processor implementation, the C3ExternalStager can supply a C3Metadata object containing a copy of the metadata retrieved from the OAI server through the parameter “md”. This is only done after successful completion of the data extraction and merging operation. Then the methods of “md” can be used to manipulate the metadata.

The manipulations of the NetCDF files are performed by calls to CDO, and if anything went wrong during them, it would be recognized by interpreting the exit status of CDO. So it can be assumed, that the requested attributes species and levels are correctly extracted. But because the requesting party usually has no information about the position of the data grid before having sent the request, the bounds of the regional and temporal requests will most probably not be exactly equalled by the attributes of the resulting data. To be able to write the precise bounding attributes to the metadata, they are read from the result file using another two CDO calls. These have been implemented in the module “netcdf_tools.py”.

Figure 7 shows the current implementation of metadata update functionality in the C3-Grid stager for the ECHAM model results. Any preview picture is removed, all but the requested tracer variables are filtered out, the horizontal bounds of the extracted region are set and the time interval is noted.

```
md.removeQuicklook()
md.filterContentInfo(self.thesaurus.translateToC3(self.requested_vars,
                                                  self.object_id))
md.setHorizontalBounds(self.lon_min, self.lon_max, self.lat_min, self.lat_max)
md.updateTimePeriod(self.timeperiod_begin_date, self.timeperiod_stop_date)
md.setObjectId(self.object_id + "." +
               datetime.datetime.utcnow().isoformat().replace(":", "-"))
md.addLineageProcessStep("c3grid extract.netcdf (CWF, ECHAM5, cdo)",
                        datetime.datetime.utcnow(),
                        self.stage_request.object_ids[0],
                        "Johannes Hendricks",
                        "http://wis.wmo.int/2006/catalogues/" +
                        "gmxCodeLists.xml#CI_RoleCode_distributor",
                        "DIR. TPA")
```

Figure 7: Metadata Update

A very important point is the creation of a *new object identifier*, because the new metadata has to be distinguishable from the original metadata, as well as clearly be associatable with the created metadata. The chosen method here is to append the current date and time to the original ID. At a later point in the project, metadata information about the aggregation of data products may help to better associate data files with each other that have been created the same way based on the same base data just at different points in time or for different users.

The last mentioned operation is the creation of processing documentation, called

“Lineage” in the ISO 19139 metadata standard. At the moment just the process is described recognizably, the instant of documentation creation, the original object ID and the responsible party. The format of the information recorded here is just following the ISO schema description and is going to be specified more strictly in the second phase of the C3-Grid to enable repetition of processes.

5.1.6.Delivering data and metadata

As long as the working path properties of the “c3env” parameter provided by the C3ExternalStager are used for temporary files and the results and if the described metadata update procedure has been used, the data provider does not have to implement anything else to have data and metadata delivered to the requested location. The external stager implementation automatically starts GridFTP transfers to accomplish this. After successful completion, all remaining data in the working directory and the working directory itself are considered to be garbage and will be deleted automatically.

5.1.7.Estimating Staging Time and File Size

StageRequests contain the property „JustAsk“, which, if set to the value „True“, requires the data provider, instead of delivering datasets, first to predict how much time the staging process would take to fulfill the given parameters and secondary, how big the data file is going to be. The scheduling system can use this information for decisions, for example about when to execute a job, that relies on the data or where to execute it, considering the amount of needed temporary storage capacity.

Therefore worst-case estimations are required, to guarantee, that the constraints the scheduling decision was made upon can always be fulfilled.

Staging Time Prediction

The staging time prediction is a task hard to do perfectly. Staging time depends on many factors, some of which, like the network between data provider and the requesting party or tape robots at the data back-end, are outside the data provider’s directly accessible system and show dynamic, through the high degree of connectivity even chaotic behavior. Even though usage statistics may give hints about the general distribution of work-load and delivery speed over times of days, underestimations cannot always be prevented.

Staging time estimation methods need to aim at compromises between achievable capacity utilization and system stability. Still statistics based heuristics are the only

practical means for optimizations here. This area will be subject to long-term observance in the C3-Grid during the following project phase to develop generic methods, if possible as well tools, to enable data providers self monitoring and development of heuristics.

At this point of development the data provider serving the CWF data is not expected to suffer from too high workloads, the base data is available on a local file system and the delivered amounts of data are usually small. Apart from that, the scheduling system currently does not weigh the estimations very high in its decisions, yet, so a pragmatic decision towards a safe static value was made.

During the test period the data was always extracted and delivered in less than 30 seconds and in most cases less than 15 seconds. For the time being the estimation value for StageTime is set to 30 seconds and will be sent to the provider web service as a property `"c3grid.StageFileResponse.StagingTime=00:00:30"`

In the data processing script `"netcdf_extraction.py"` this is implemented as the function shown in .

```
def estimateProcessingTime(self) :
    self.stage_request.respondStageTime("00:00:30")
    self.stage_request.respondWarning("Warning: %r does only implement \
static processing time prediction!" % self.__class__)
```

Figure 8: Processing Time Estimation

File Size Prediction

File size prediction is a comparably simple task for the case of CWF data delivery, because no compression is used. The file size linearly depends on the number of requested species, pressure levels and region extents.

The gaussian grid of the base data has a resolution of 128 gridpoints along the longitude and 64 gridpoints along the latitude on each level and for each species. There are 8 data sets per day in time intervals of 3 hours. The format of the data values is 32-bit Float. So the file size calculates as displayed in Figure 8.

„lon_range“ and „lat_range“ contain the width of the region along the longitude and latitude axes in degrees. So the product of `"gp_lon"` and `"gp_lat"` is the estimated number of contained grid points within the region. After multiplication of the grid point size with the number of requested species and `time_steps`, the `"clear_size"` is the amount of data in bytes per level. The current extraction method does not automatically determine levels to retrieve between the minimum and maximum pressure level so the number of levels is either one, in the case that maximum and minimum level constraints are equal, or two levels are calculated. The result is increased by a fix number of Bytes

for data overhead for the structure of the NetCDF file, variable names and contained comments. Tests show, that 4000 Bytes are reasonable.

```
gp_lon = (lon_range * BASEDATA_RES_LON / 360.0) + 1
gp_lat = (lat_range * BASEDATA_RES_LAT / 180.0) + 1
time_period = self.timeperiod_stop_date - self.timeperiod_begin_date
time_steps = int ( (time_period.hours / 24.0 + time_period.days) * \
    BASEDATA_RES_DAY + 1 )
size_clear = gp_lon * gp_lat * len(self.requested_vars) * time_steps \
    * BASEDATA_BYTES_PER_GP
if self.alt_min != self.alt_max:
    size_clear *= 2
estimated_size = size_clear + 4000 # estimated overhead
self.stage_request_responseRequiredSize(estimated_size)
```

Figure 8: File Size Calculation

The result is very close to the sizes of the actually generated files and just a little bit bigger. The estimation error diminishes when the selected region grows.

The size estimation result is transmitted to the web service automatically by the stager via the property "*c3grid.StageFileResponse.RequiredSize*" when issued with the method "*respondRequiredSize*".

The estimation method could be increased in versatility by retrieving the resolution constants from the base data. This was left out to reduce the need of accessing the data for estimation and thus increasing speed especially when the data is momentarily swapped to a long term storage device.

5.2. Visual Plot with GMT and Ghostscript

While from the data staging prototype only the core functionality has been kept, but translated into Python to benefit from the PyModEST library, the visualization script "plot_netcdf.sh" (G1) has been maintained to a large degree. The implementation as a script that only must be executable by the C3-Grid user on a Globus Toolkit 4 accessible resource, made it more independent of C3-Grid specific environment from the start.

It benefits from the C3-Grid standard environment requirements, though. The script has been decoupled from the data provider by removing the ini-file parameterization and now is able to update the metadata of the input file. The source code is available in appendix 7.2.

5.2.1. Features and Requirements

The script takes a gridpoint formatted NetCDF file and optionally a metadata file as inputs and creates a PNG image containing a colored plot of the data, laid out with a color bar and textual annotations. If a metadata file is given, it is copied to the same folder the image is written into and updated.

In addition to the previous requirements

- *Generic Mapping Tools*¹¹ >= V4.2.0 with *NetCDF* support,
- *GhostScript*¹², implicitly used by GMT,
- *Bash*, *AWK*, *md5sum* (Unix/Linux standards),
- *Environment Modules*¹³,

decoupling from the data provider ini-file now adds as well

- *Climate Data Operators*¹⁴ (CDO)

to the list, which enable the script to read all necessary parameters directly from the data file.

The script now accepts the following parameters:

`-i <input NetCDF data-file>` (required):

The file containing the gridpoint formatted data to be plotted

`-m <iso metadata file>` :

Used to create updated metadata for the image

`-o <output file path>` (required):

The full path to the PNG image to create. A metadata file is going to be created in the same directory, if “-m” is given.

`-t <temporary directory>`:

Within the temporary directory an individual subdirectory is created for each plot process with a different result image name. If “-t” is not given, the script tries to find the C3-Grid environment variable `$C3_WN_SCRATCH` as replacement or uses the current directory, if even that fails.

For the simplicity of this interface a small price has to be paid. The order of the species, levels and timesteps in the data file matters. Only the first species, level and timestep found by CDO in the file is plotted. This limitation is not so very difficult to overcome, by adding more parameters, but so far it is exactly what is needed to realize the workflow as straight-forward as possible and with little code needed to maintain consistency. The user of the workflow is supposed to select exactly one region, pressure

¹¹ Generic Mapping Tools: <http://gmt.soest.hawaii.edu/>

¹² GhostScript: <http://www.ghostscript.com/>

¹³ Environment Modules: <http://modules.sourceforge.net/>

¹⁴ Climate Data Operators: <http://www.mpimet.mpg.de/fileadmin/software/cdo/>

level and species to be plotted at one moment in time, so the order issue will never arise anyway.

The following section describes the implementation.

5.2.2. Plot and Layout of NetCDF Data

The script “plot_netcdf.sh” is divided into seven phases:

1. Configuration and Default Values
2. Interpretation of Parameters and Input Files
3. Layout Phase
4. Preparation Phase
5. Rendering Phase
6. Metadata Phase
7. Clean-Up Phase

Phase 1: Configuration

The script first initializes the environment by loading the environment modules for C3-Grid, GMT and CDO, before standard layout parameters are set.

For many calculations AWK is used for its floating-point support. It is important to export the environment variable “LC_NUMERIC=C” to keep AWK from using commas instead of decimal points in some locales, what otherwise would produce a mess in some parameterizations.

Most of the other default values serve the layout features and are described further down this document.

Phase 2: Interpretation of Parameters and Input Files

At the beginning of this phase, the command line parameters are analyzed and the given input files are checked for existence, so that any missing input files or forgetting to specify the output file lead to an early abortion of the script.

Then through several calls of CDO, the relevant data properties, like date and time step, geographic region, species, altitude level in Pascal are read from the NetCDF file. All of this depends on the textual output format of CDO and adjustments may needed for future versions of CDO. The tests run happily with CDO version 1.0.7 and 1.0.8.

The MD5 checksum of the output file path is used to create a unique working

directory under the temporary files directory. So it is a requirement to the scheduling system to use unique names for every image requested, to prevent clashes of plotting processes running in parallel.

Phase 3: Layout

The image to be produced consists of the data plot, the color bar and several annotations as can be seen in Figure 9. After all label texts have been determined and the geographical dimensions of the map have been analyzed, in this phase the adaptive layout parameters are calculated, so even under extreme aspect ratios of longitudinal length and latitudinal width of the map, the additional information always fits with it properly into a maximum height of 830 pixels and width of 600 pixels. That is suitable for display in a web browser.

The values correspond to the layout defaults mentioned in Phase 5.2.2. The contents are rendered at 96 dpi resolution on a virtual DIN A4 sheet in a frame of maximum 16 cm width and 22 cm height.

The map is decorated with tick marks in visually pleasing distances and value step widths. This is calculated by trying in a first step to fit nine marks in equal distances along the longer edge of the map. The distance factor is transformed to exponential notation with a rounded mantisse of one digit length before the dot. So now the mantisse is guaranteed to have a value between 1 and 9. If the mantisse is 1 or 2, the exponential number now represents the width of major tick marks and the minor tick marks are set to 0.5 with the same exponent. One example would correspond to a combination of $10^{\circ}/5^{\circ}$, as in Figure 9. A mantisse between 3 and 9 is leads to a minor tick mark distance that is the exponential value with a new mantisse of the integer quotient of the mantisse and 3. Then every third minor tick mark distance a

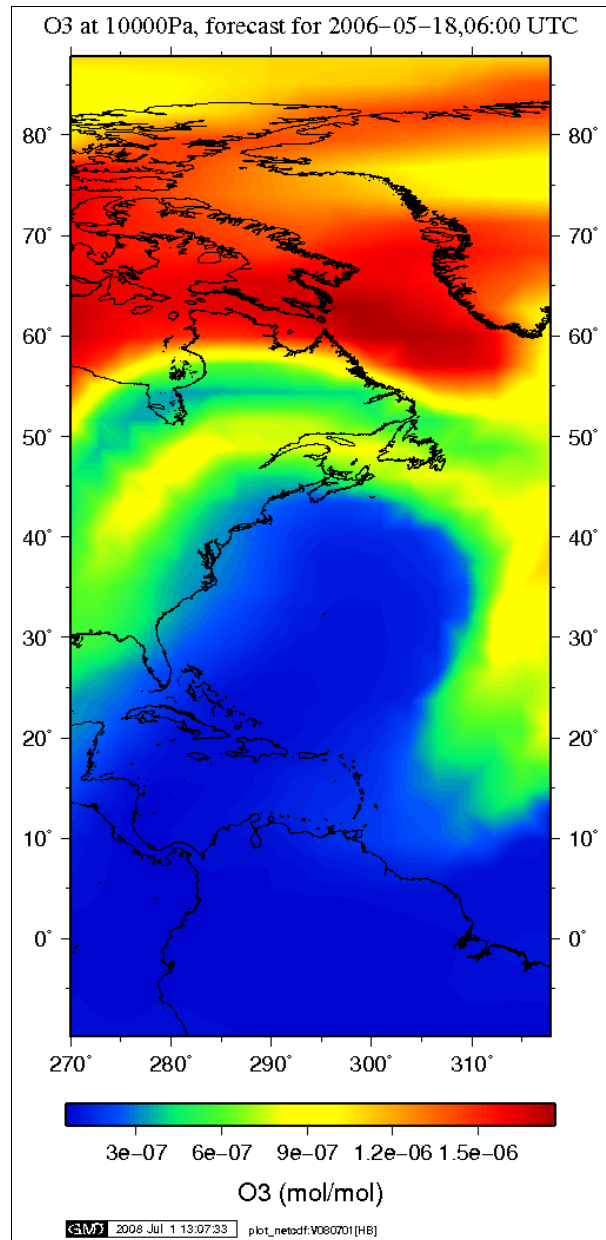


Figure 9: Output PNG of plot_netcdf.sh

major tick mark is placed (example: 30°/10°).

The color bar is scaled to optimally cover the range from the minimum to the maximum value of the displayed map. All elements other than the date stamp at the bottom and the plot_netcdf version information next to it are horizontally centered.

Phase 4: Preparation

Now the individual temporary directory and the target directory, should it not already exist, are created and the GMT are configured using the tool “gmtset” to work towards the chosen document format and font sizes.

After the color palette file has been generated with “makecpt” from the minimum and maximum values of the species data set, the color bar tick mark distance is calculated with a similar but simpler method as for the map annotations.

Phase 5: Rendering

The pre-calculated values and settings are now used to generate the postscript file through several consecutive tool calls. The species data is extracted from the NetCDF file to an ASCII file with “grd2xyz”, then projected and triangulated using the palette file. Over this visual representation of the data, coastlines and map annotations are painted with “pscoast”. The headline is placed at the designated position with “pstext” and the color bar with “psscale”. “ps2raster” finally creates the PNG file that is trimmed to the painted content and simply moved to the output file position.

Phase 6: Metadata

If a metadata file has been specified and the file can be found, an update is performed as described in the next section. The updated metadata is placed in the same directory as the created PNG image.

Phase 7: Clean-Up

At the end all temporary files, GMT settings and sub-directories are removed from the temporary directory.

5.2.3. Metadata Update

Metadata is generated for an image put out by plot_netcdf.sh only if a metadata file for the input data has been specified with the parameter “-m”. This takes the role of the metadata template in an update process using the same method as described for

the creation of metadata for the CWF data results in section 4.2. The XSL transformation in this case

- removes all species but the plotted one from the metadata,
- updates the distribution info (file type and size),
- appends the lineage section with a description of the used tools and
- creates a new identifier

5.2.4.Extensibility

The `plot_netcdf.sh` script is extensible in many ways to fulfill future demands.

Throughout the whole script tool output to `stdout` is either directly used or redirected to `stderr` to keep the `stdout` of the script clear for eventual communication with the calling software.

The effort to implement more command line option controllable features, for example for species or time step selection, external control of layout and output parameters is considerably low, for example to directly deliver the postscript file instead of the PNG which is usually generated from it.

6. Conclusion

We have described the fundamental work on a scientific workflow involving data, model and diagnosis components that is necessary for their implementation on the C3-Grid. The main aspects to handle are:

- Components the workflow is composed of. Some properties as guides for modularization are:
 - Interactions required with the C3-Grid environment (data providers, portal, metadata, file systems and environment settings)
 - Runnability on more than one compute resource
 - Reusability in different workflows.
- Data that the users want to handle on the portal or that is transmitted between components. Things to accomplish are:
 - Installation of C3 data provision services (C3ProviderTemplate), if not already available and extendable, for all data that has to be explicitly visible to the users.
 - Creation of metadata descriptions for all such data.
 - Implementation of data type and storage method specific external staging scripts as adaptors between data provider web-services and storage resources.
 - Implementation of metadata update processes for all components.

The original workflow “Measurement Campaign Support” as shown in Figure 1 in section 2.5 is reflected in the current implementation for the C3-Grid as in Figure 10. The “Chemical Weather Forecast Data Production” contains pre-processing, simulation and pos-processing at the moment, because only the results come in contact with C3-Grid components. The structure of the “Visualization Workflow” implementation is very close to the original, because it benefits directly from the extraction model of the C3-Grid data provider.

One difference apart from the composition is the static implementation of layout parameters. One reason for this was getting round the creation of a specific user interface for layout control at this point. The regional data selection interface of the portal “Download Assistant” is currently sufficient for that purpose.

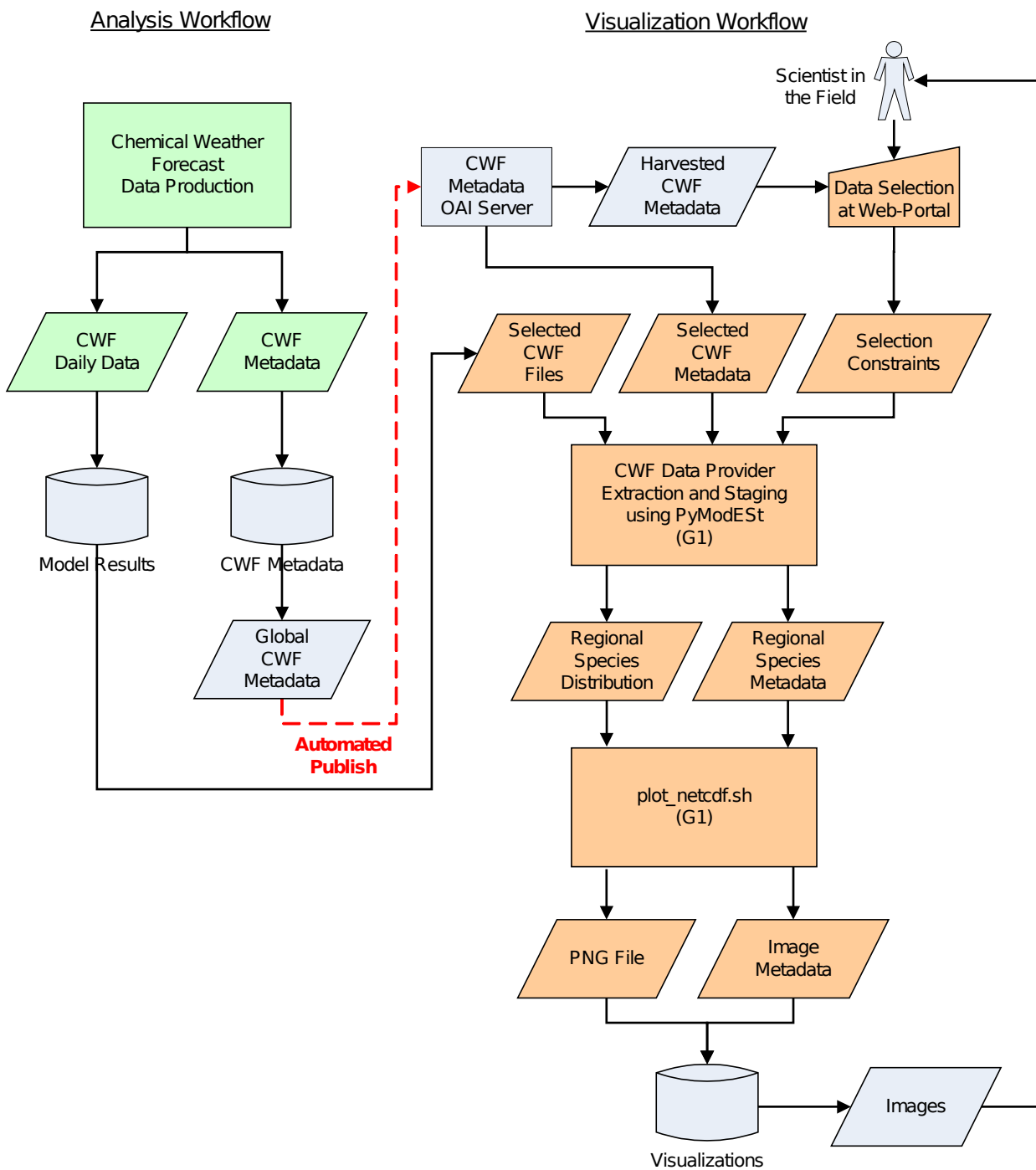


Figure 10: C3-Grid Implementation of the „Measurement Campaign Support“

One C3-Grid data management feature yet missing for completion of the workflow application is the “Automated Publish” function marked red in the figure. This is currently under development and allows resulting data to be become searchable on the portal by automatically placing the metadata in an OAI server. Currently this has to be done manually.

As we have seen, for the tasks that have to be adressed to deal with all aspects of workflow implementation on the C3-Grid, there is no single sufficient method of

implementing. In many cases a trade-off between flexibility of components, transparency of system or use specific implementations, complexity of component interactions and data specific requirements has to be made, so that a workflow that is already manually runnable independently of the C3-Grid, can efficiently be transferred to the C3-Grid with reasonable effort.

Currently especially metadata handling, data provision and making own tools runnable in a grid environment create considerable overhead for scientists, who expect to get an easy to use toolkit and computing resources through the C3-Grid and who know on the other hand, how to sign-in on a remote system with SSH and start their tools manually. It seems that it is not the creator of the data or the implementer of workflows who benefits from the additional work, but users from other institutions or later projects.

From that point-of-view, workflow and data providers have to recognize the worth of this effort not only for commercial applications, but as well as means of scientific publication that significantly increase the visibility, traceability and usability of their work and research results.

7. Appendix

From all source codes any change comments and uncommented code have been removed for this document.

7.1. NetCDF Stager (G1, based on PyModESt)

7.1.1. Stager Starting Script: ipa_c3stager_mod.py

```
#!/usr/bin/python
#
# ipa_c3stager_mod.py
#
# Version: 0.6 (alpha)
# Notes:   C3-Grid G1 ready, modularized
#
# Created: 21.04.2008 berg_hn <henning.bergmeyer@dlr.de>
#
# Copyright (C) 2007 DLR/SISTEC, Germany
#
# http://www.dlr.de/sc
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

from c3grid.c3datapvider.c3stager import C3ExternalStager
import netcdf_extraction

PROCESSOR_TYPES = {"de.dlr.ipa.CWF" : netcdf_extraction.IPA_NCDF_Processor}

if __name__ == "__main__":
    C3ExternalStager(PROCESSOR_TYPES)
```

7.1.2. Data Processor for CWF-NetCDF Files: netcdf_extraction.py

```
#!/usr/bin/python
#
# netcdf_extraction.py
#
# Version: 0.3 (alpha)
# Notes:   C3-Grid G1 ready
#
# Created: 04.02.2008 berg_hn <henning.bergmeyer@dlr.de>
# Changed: 10.04.2008 berg_hn <henning.bergmeyer@dlr.de>
#
```

```

# Copyright (C) 2007 DLR/SISTEC, Germany
#
# http://www.dlr.de/sc
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
"""
In this module a data processor for the data provider for data from the IPA
ECHAM/MESSy repository is realized.

Non-standard library dependencies:

    * iso8601 (http://pypi.python.org/pypi/iso8601)
    * pyhdf (http://pysclint.sourceforge.net/pyhdf/)

@todo: update vertical bounds in meta data
"""

import datetime
import logging
import os
import urllib
import iso8601
import Numeric
#import tempfile
import subprocess
import netcdf_tools

from c3grid.c3dataprovider.c3thesaurus import C3Thesaurus
#from c3grid.c3dataprovider.c3metadata import C3Metadata
from c3grid.c3dataprovider.c3exceptions import StagerException
from c3grid.c3dataprovider.c3dataprocessing import C3DataProcessor
from c3grid.c3dataprovider.c3dataprocessing import SUFFIX_NETCDF
#from c3grid.c3dataprovider.gaussgrids import RegularGaussianGridHelper

#
# Scope-keyed dictionary of dictionaries that map C3-Grid attribute names (keys)
# on-to-one to local data attribute names (values).
#
C3_CWF_SCOPE_MAP = \
    { "de.dlr.ipa.ECHAM5" : \
      { "mole_fraction_of_carbon_monoxide_in_air" : "CO", \
        "mole_fraction_of_carbon_dioxide_in_air" : "CO2", \
        "mole_fraction_of_hydroperoxy_radical_in_air" : "HO2", \
        "mole_fraction_of_hydroxyl_radical_in_air" : "OH", \
        "mole_fraction_of_nitrogen_dioxide_in_air" : "NO2", \
        "mole_fraction_of_nitrogen_oxide_in_air" : "NO", \
        "mole_fraction_of_ozone_in_air" : "O3", \
        "mole_fraction_of_peroxy_acetyl_nitrate_in_air" : "PAN", \
        "geopotential" : "geosp", \
        "surface_air_pressure" : "aps" \
      }
    }

```

```

        } \
    }

#
# native file format
#
NATIVE_FORMAT_SUFFIX = SUFFIX_NETCDF

class IPA_NCDF_Processor(C3DataProcessor) :
    """
    This class realizes the data specific extraction process from the source on
    the web.

    Values that have to be adjusted to fit to the data are copied into a local
    attribute.
    """

    ENABLE_CONVERSION = False

    def __init__(self, c3env, stage_request) :
        """
        Initializes the data processor by reading the stage request and adjusting
        to the local environment.

        @type c3env: C3Environment
        @param c3env: It is stored an later used to determine file locations and
        execution environment
        @type stage_request: C3StageRequest
        @param stage_request: Stage request that provides the parameters for
        processing
        """

        C3DataProcessor.__init__(self, c3env, stage_request)

        #
        # Initialize thesaurus supporting the scope
        #
        self.thesaurus = C3Thesaurus(C3_CWF_SCOPE_MAP)

        #
        # This processor will only work on the first element of the object list
        and ignore any more entries.
        #
        self.object_id = self.stage_request.object_ids[0]

        #
        # Make local copies of the requested region bounds. These will later be
        adjusted to the real bounds of
        # the resulting data for a precise update of the metadata
        #
        if self.stage_request.alt_unit != "hPa" :
            raise StagerException("Altitude unit \"%s\" is not supported by the
            data set" % self.stage_request.alt_unit)
        self.alt_min = float(self.stage_request.alt_min)*100
        self.alt_max = float(self.stage_request.alt_max)*100
        self.lon_min = self.stage_request.lon_min
        self.lon_max = self.stage_request.lon_max
        self.lat_min = self.stage_request.lat_min
        self.lat_max = self.stage_request.lat_max

```

```

self.timeperiod_begin_date = self.stage_request.time_min
self.timeperiod_stop_date = self.stage_request.time_max

#
# Variable mapping: The C3/CF-Names of the variables are translated to
the local namespace
#
self.cf_list, missing =
self.thesaurus.stripMissingEntriesInC3Scope(self.stage_request.cf_list,
self.object_id)
self.requested_vars = self.thesaurus.translateFromC3(self.cf_list,
self.object_id)
if len(missing) > 0 :
    logging.warning("some requested variables are not defined in the
thesaurus of this provider and will be ignored: %s" % missing)
    logging.warning("delivering cf variables %s" % self.cf_list)
if len(self.requested_vars) == 0:
    raise StagerException("WARNING: The data set %r does not contain any
of the requested variables %r!" \
        % (self.object_id, self.stage_request.cf_list))

def retrieveAndFilterDataFiles(self):
    """
    Retrieves the base datafiles, filters them according to the self and
merges them in a result file.
    To keep local storage space requirements low, retrieving and filtering is
done for one base file at a time.

    The underlying base data is sliced into monthly files, that contain 2D
georeferenced data for each variable.
    Each requested variable data set in the target file is extended with a
third dimension "time", along which the extracts
from the monthly files are placed.

    There is no failsafe error handling at the moment. If reading any base
datafile fails, the complete process is going to fail.
    """

#
# calculate parameters (region, vars, levels, first and last time_step)
#
SOURCE_FILE_NAME_DATE_PATTERN = "forecast_%Y%m.%d_tracer_gp.nc"
SOURCE_PATH = "/data/%s"
TIME_STEP_LENGTH_HOURS = 3
TIME_STEPS_PER_DAY = int(24 / TIME_STEP_LENGTH_HOURS)

CDO_CMD = "cdo"
CDO_TIMESTEP_PAR = "-seltimestep"
CDO_MERGE_PARS = "mergetime"

gp_pars = "sp2gp"
var_pars = ','.join(["-selvar"] + self.requested_vars)
region_pars = ','.join(["-sellonlatbox", str(self.lon_min),
str(self.lon_max), str(self.lat_min), str(self.lat_max)])
if self.alt_min == self.alt_max :
    pressure_level_pars = ','.join(["-ml2pl", str(self.alt_min)])
else :
    pressure_level_pars = ','.join(["-ml2pl", str(self.alt_min),
str(self.alt_max)])

```

```

        timestep_start = Numeric.ceil((self.timeperiod_begin_date.hour +
self.timeperiod_begin_date.minute / 60 + self.timeperiod_begin_date.second /
3600) / TIME_STEP_LENGTH_HOURS)
        timestep_stop = Numeric.floor((self.timeperiod_stop_date.hour +
self.timeperiod_stop_date.minute / 60 + self.timeperiod_stop_date.second / 3600)
/ TIME_STEP_LENGTH_HOURS)

#
# Loop over all days within the requested time period.
# For each day the base data file is downloaded,
# the necessary data extracted and packaged into target file.
#
day_step = datetime.timedelta(days=1)
date = self.timeperiod_begin_date.date
while date <= self.timeperiod_stop_date.date :

    #
    # Retrieve a base data file from the server
    #
    src_file_name = date.strftime(SOURCE_FILE_NAME_DATE_PATTERN)
    src_url = SOURCE_PATH % src_file_name
    try:
        logging.debug("retrieving file %s" % (src_url, src_file_name))
        fn, headers = urllib.urlretrieve(url=src_url,
filename=self.c3env.workdir + src_file_name)
        logging.debug("file %s retrieved" % fn)
    except :
        msg = "ERROR: missing data file of date: %s" % date.isoformat()
        raise StagerException(msg)

    #
    # Determine timesteps to extract
    #
    if date == self.timeperiod_begin_date.date : # First day
        startstep = timestep_start
    else :
        startstep = 0
    if date == self.timeperiod_stop_date.date : # Last day
        stopstep = timestep_stop
    else :
        stopstep = TIME_STEPS_PER_DAY - 1
    time_steps_str = ",".join([CDO_Timestep_PAR] + [str(z) for z in
range(startstep, stopstep + 1)])

    #
    # Convert file from spectral to grid-point data and extract only
    requested vars at requested time steps
    #
    cdo_call = " ".join([CDO_CMD, gp_pars, var_pars, time_steps_str, fn,
self.c3env.localtempfilepath])
    retcode = subprocess.call(";".join(["module load c3grid cdo",
cdo_call]), shell=True)
    if retcode < 0:
        raise StagerException, "CDO (vars, time) was terminated by signal
%i" % -retcode
    os.remove(fn)

    #
    # Extract requested region, interpolate model level to pressure level
    and select the pressure levels
    #

```

```

        cdo_call = " ".join([CDO_CMD, region_pars, pressure_level_pars,
self.c3env.localtempfilepath, fn])
        retcode = subprocess.call(";".join(["module load c3grid cdo",
cdo_call]), shell=True)
        if retcode < 0:
            raise StagerException, "CDO (region, pressure level) was
terminated by signal %i" % -retcode
            os.remove(self.c3env.localtempfilepath)

#
# Merge extract with previous extracts or create initial basedatafile
#
        if os.path.exists(self.c3env.localbasefilepath) :
            os.rename(self.c3env.localbasefilepath,
self.c3env.localtempfilepath)
            cdo_call = " ".join([CDO_CMD, CDO_MERGE_PARS, fn,
self.c3env.localtempfilepath, self.c3env.localbasefilepath])
            retcode = subprocess.call(";".join(["module load c3grid cdo",
cdo_call]), shell=True)
            if retcode < 0:
                raise StagerException, "CDO (mergetime) was terminated by
signal %i" % -retcode
                os.remove(self.c3env.localtempfilepath)
            else :
                os.rename(fn, self.c3env.localbasefilepath)

#
# Continue with next day
#
        date += day_step

#
# update the internal representation of data set properties
#
        (self.lon_min, self.lon_max, self.lat_min, self.lat_max) =
netcdf_tools.getRegionalBoundsFromNetCDF(self.c3env.localbasefilepath)
        (self.timeperiod_begin_date, self.timeperiod_stop_date) =
netcdf_tools.getTimePeriod(self.c3env.localbasefilepath)

#
# when the data is asked to be provided in other file formats, commit the
conversion now
#
        if self.ENABLE_CONVERSION :
            logging.warning("No file format converters implemented!")
        else :
            # logging.warning("file format conversion is disabled")
            if not (self.stage_request.targetfileformat.lower() in
NATIVE_FORMAT_SUFFIX) :
                logging.warning("File format conversion is disabled but file is
asked to be converted to " + self.stage_request.targetfileformat.lower())

def updateMetadata(self, md) :
    """
    Reflect the processing of the data in the metadata.
    @type md: C3Metadata
    @param md: The C3Metadata instance to update.
    @todo: Update vertical bounds
    """

    md.removeQuicklook()

```

```

        md.filterContentInfo(self.thesaurus.translateToC3(self.requested_vars,
self.object_id))
        md.setHorizontalBounds(self.lon_min, self.lon_max, self.lat_min,
self.lat_max)
        md.updateTimePeriod(self.timeperiod_begin_date,
self.timeperiod_stop_date)

        md.setObjectId(self.object_id + "." +
datetime.datetime.utcnow.isoformat().replace(":", "-"))
        md.addLineageProcessStep("c3grid extract.netcdf (CWF, ECHAM5, cdo)",
datetime.datetime.utcnow(), \
self.stage_request.object_ids[0], \
"Johannes Hendricks", \
"http://wis.wmo.int/2006/catalogues/gmxCodelists
.xml#CI_RoleCode_distributor", \
"DLR, IPA")

def estimateFileSize(self) :
    BASEDATA_RES_LON = 128
    BASEDATA_RES_LAT = 64
    BASEDATA_RES_DAY = 8
    BASEDATA_BYTES_PER_GP = 4

    while self.lon_min < 0 :
        self.lon_min += 360.0
    self.lon_min %= 360.0
    while self.lon_max < 0 :
        self.lon_max += 360.0
    self.lon_max %= 360.0
    lon_range = self.lon_max - self.lon_min
    if lon_range < 0 :
        lon_range = 360.0 + lon_range

    lat_range = self.lat_max - self.lat_min
    if lat_range < 0:
        lat_range = -lat_range

    gp_lon = (lon_range * BASEDATA_RES_LON / 360.0) + 1
    gp_lat = (lat_range * BASEDATA_RES_LAT / 180.0) + 1
    time_period = self.timeperiod_stop_date - self.timeperiod_begin_date
    time_steps = int ((time_period.hours / 24.0 + time_period.days) *
BASEDATA_RES_DAY + 1)

    size_clear = gp_lon * gp_lat * len(self.requested_vars) * time_steps *
BASEDATA_BYTES_PER_GP
    if self.alt_min != self.alt_max:
        size_clear *= 2
    estimated_size = size_clear + 40000 # estimated overhead
    self.stage_request.respondRequiredSize(estimated_size)

```

7.1.3. NetCDF Tool Methods Module: netcdf_tools.py

```

#!/usr/bin/python
#
# netcdf_tools.py
#
# Version: 0.1 (alpha)
# Notes:   C3-Grid G1 ready
#
# Created: 01.07.2008 berg_hn <henning.bergmeyer@dlr.de>
# Changed: 01.07.2008 berg_hn <henning.bergmeyer@dlr.de>

```

```

#
# Copyright (C) 2007 DLR/SISTEC, Germany
#
# http://www.dlr.de/sc
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
"""
This module serves as a collection for NetCDF operations using CDO as a
subprocess.

Dependencies:
    Module Environment
    CDO + CDO environment module
    ISO8601 Python library
"""

import subprocess
import iso8601

def getRegionalBoundsFromNetCDF(filename) :
    """
    This method returns the regional bounds of the dataset in a NetCDF file as a
    tuple.
    @param filename: A path to a NetCDF file.
    @type filename: String
    @return: Tuple of gaussian geographic bounds (min_lon, max_lon, min_lat,
    max_lat)
    @rtype: (float, float, float, float)
    """

    CDO_CMD = "module load cdo ; cdo"
    output = subprocess.Popen([CDO_CMD, "griddes", filename],
    stdout=PIPE).communicate()[0]
    ncdf_props = output.split()

    xsize = ncdf_props[ncdf_props.index("xsize") + 2]
    xinc = ncdf_props[ncdf_props.index("xinc") + 2]
    xfirst = ncdf_props[ncdf_props.index("xfirst") + 2]
    xlast = xfirst + xinc * (xsize - 1)

    if xlast > 360.0 :
        xlast %= 360.0

    yfirst = ncdf_props[ncdf_props.index("yvals") + 2]
    ylast = ncdf_props[-1]

    if yfirst > ylast :
        yfirst, ylast = ylast, yfirst

    return (float(xfirst), float(xlast), float(yfirst), float(ylast))

```

```

def getTimePeriod(filename) :
    """
    @return: (start_date, stop_date)
    @rtype: (datetime, datetime)
    """

    CDO_CMD = "module load cdo ; cdo"
    output = subprocess.Popen([CDO_CMD, "showtime", filename],
    stdout=PIPE).communicate()[0]
    ncdf_props = output.split()

    return (iso8601.parse_date(ncdf_props[0]+' :00'),
    iso8601.parse_date(ncdf_props[-1]+' :00'))

```

7.2.plot_netcdf.sh (G1)

```

#!/bin/bash
# for identification of script version and association with generated picture
# Please update after any change
PLOT_NETCDF_VERSION="plot_netcdf:V080701[HB]"

#####
### Script: plot_netcdf_potsdam.sh
### Authors: Henning Bergmeyer (DLR SC-VK) [HB]
###          Christian Grimme (Uni-DO) [CG]
###          Christian Kurz (DLR IPA) [CK]
###
### Creation:2007-05-31
### Modified:2008-07-01
### Status: C3-Grid Generation 1, alpha
###
### This performs the plot of a netcdf file containing an extract of
### analysis results of the grid workflow "Chemical Weather Forecast".
###
### Requirements:
### CDO installed with netCDF support
### GMT installed with netCDF support
### Ghostscript
### awk
### md5sum
### C3Grid environment
### * directories (scratch directory)
### * environment variables
### ($C3_WN_SCRATCH, $C3_WN_WORKSPACE, $C3_WN_HTDPCS, $PATH)
### * standard tools (Environment Modules)
###
### Parameters:
### -i <input data-file> (required)
### The file containing netCDF data for the plot
### -m <metadata-file>
### An xml file containing metadata for the input dataset.
### If given it is used to create metadata for the created
### picture
### -o <outfile-path>
### Path to the PNG file for the resulting image
### -t <temporary-directory>
### Path to a directory with write access to temporarily store files in
###
###

```

```

### Output: a PNG image. The process creates temporary files in the scratch
space. On
###      successful completion those are removed automatically.
###

echo "*****" 1>&2
echo "** Plot of netCDF-Files for the Chemical Wheather Forecast" 1>&2
echo "*****" 1>&2

export PATH

if [ -r "/etc/profile.d/modules.sh" ]; then
    . /etc/profile.d/modules.sh
    module load c3grid gmt cdo 1>&2
else
    echo $0: /etc/profile.d/modules.sh not found 1>&2
    exit 1
# . init_dlr.sh # just for local tests at IPA
fi

#####
### Default Values
#####

# Coding of floating point values
export LC_NUMERIC=C

# Plot Parameters
# Image resolution in dpi. Higher values increase raster image dimensions and
# improve readability of text.
PNG_RESOLUTION=96
# Average value of tick marks on the longer edge of the map
MAP_TICK_MARKS=9

# empirical values for A4 portrait plots
MAX_HEIGHT=22
MAX_WIDTH=16
MIN_COLORBAR_WIDTH=11
Y_SHIFT=4.75

PAPER=a4
ORIENTATION=portrait
ANNOTATION_FONT_SIZE=14p
LABEL_FONT_SIZE=16p

TIME_RESOLUTION=3 #time resolution [hours] in input file

#####
### Interpretation Phase
#####

# Read command line parameters
# interpret file locations relative to c3grid environment
while getopts i:m:o:t: Option; do
    case $Option in
        i) INFILE=$OPTARG ;;
        m) METAFILE=$OPTARG ;;
        o) OUTFILE=$OPTARG ;;
        t) TEMP_DIR=$OPTARG ;;
        *) echo "WARNING: unknown option "$Option 1>&2 ;;
    esac

```

```

done

# There is a file needed, that provides the data to be plotted
# Only proceed if a data file was specified
if [ -z $INFILE ] ; then
    echo "*** ERROR: Data file not specified! Use parameter -d <file-path>" 1>&2
    echo "*** Aborting..." 1>&2
    exit 1
else
    echo "*** Data file:" $INFILE 1>&2
fi

# Only proceed if the data file exists
if [ ! -e $INFILE ] ; then
    echo "*** ERROR: Data file does not exist!" 1>&2
    echo "*** Aborting..." 1>&2
    exit 1
fi

# The output is to be stored at a specific position
# Only proceed if an output file was specified
if [ -z $OUTFILE ] ; then
    echo "*** ERROR: Output file not specified, Use parameter -o <png-file-
path>" 1>&2
    echo "*** Aborting..." 1>&2
    exit 1
else
    echo "*** Output file:" $OUTFILE 1>&2
fi

# Check if the output file already exists and if so cast a warning
if [ -e $OUTFILE ] ; then
    echo "*** WARNING: Target file does already exist and will be overwritten!"
1>&2
fi

# If no directory for temporary files is given, work in the C3-Grid Scratch or
the current directory
if [ -z $TEMP_DIR ] ; then
    if [ -z $C3_WN_SCRATCH ] ; then
        TEMP_DIR=$PWD
    else
        TEMP_DIR=$C3_WN_SCRATCH
    fi
fi

# Read parameters from the INFILE using CDO
# get species
VarList=$(cdo showvar $INFILE)
var_index=${#VarList[*]}

# get time step
TimeConstraint=$(cdo showtime $INFILE)
Date=$(cdo showdate $INFILE)

# determine plot region
xsize=$(cdo griddes $INFILE | grep 'xsize' )
xsize=${xsize[2]}
xinc=$(cdo griddes $INFILE | grep 'xinc')
xinc=${xinc[2]}
xfirst=$(cdo griddes $INFILE | grep 'xfirst')
xfirst=${xfirst[2]}

```

```

xlast=$(awk "BEGIN { print( ("xfirst" + "xinc" * ("xsize" - 1)) ) }")

if [[ "1" == $(awk "BEGIN { print( "xlast" > 360 ) }") ]] ; then
    xlast = $(awk "BEGIN { print ( "xlast" % 360.0 ) }")
fi

ysize=$(cdo griddes $INFILE | grep 'ysize')
ysize=${ysize[2]}
yvals=$(cdo griddes $INFILE | grep 'yvals')
yfirst=${yvals[2]}
des=$(cdo griddes $INFILE)
ylast=$(awk "BEGIN { print(("#{@des[*]}" - 1) ) }")
ylast=${des[$ylast]}

SpaceConstraint[0]=$xfirst
SpaceConstraint[1]=$xlast
if [[ "1" == $(awk "BEGIN { print( "yfirst" > "ylast" ) }") ]] ; then
    SpaceConstraint[2]=$ylast
    SpaceConstraint[3]=$yfirst
else
    SpaceConstraint[2]=$yfirst
    SpaceConstraint[3]=$ylast
fi

# Take only the first altitude level.
AltitudeConstraint=$(cdo showlevel $INFILE)
AltitudeConstraint=${AltitudeConstraint[0]}
AltitudeUnit="Pa"

# Extract outfile name, target directory and WID
OUTFILENAME=${OUTFILE##*/}
TGTDIR=${OUTFILE%/*}
WID=$(echo $OUTFILE | md5sum)
WID=${WID[0]}

# Declare filenames of temporary files
SCRATCHDIR=$TEMP_DIR/$WID
TMPPS=$SCRATCHDIR/tmp.ps
PALFILE=$SCRATCHDIR/tmp.cpt

if [ "$var_index" == "0" ] ; then
    echo "*** ERROR: No plottable species specified!" 1>&2
    echo "*** Aborting..." 1>&2
    exit 1
fi

#####
### Layout Phase
#####

# Only the first species in the list of variables is going to be plotted
Species=${VarList[0]}

#Unit="0@-3@- (mol/mol)"
Unit=$Species" \ (mol/mol\)"
#HeadlineText="0@-3@- at 200hPa, forecast for 2006/05/18 18UTC"
#Date=${TimeConstraint[0]}
HeadlineText=$Species" at "${AltitudeConstraint[0]}$AltitudeUnit", forecast for
"$TimeConstraint" UTC" #HB 20080701 correct time again

# PROJECTION and position of title

```

```

CentralMeridian=$( awk "BEGIN { print(("${SpaceConstraint[0]}" +
"${SpaceConstraint[1]})/2); }")
HeadlinePosition=${CentralMeridian}" "${SpaceConstraint[3]}
PROJECTION="Q"${CentralMeridian}

# parameter string resembling the campaign region to be plotted
PlotRange=${SpaceConstraint[0]}/${SpaceConstraint[1]}/${SpaceConstraint[2]}/
"${SpaceConstraint[3]}

# interval sizes of longitude and latitude
LON_WIDTH=$( awk "BEGIN { printf( "${SpaceConstraint[1]}" -
"${SpaceConstraint[0]}" ); }")
LAT_HEIGHT=$( awk "BEGIN { printf( "${SpaceConstraint[3]}" -
"${SpaceConstraint[2]}" ); }")

# for the plot not to be cut by page borders it must be scaled to fit
# perform scaling considering the relation of width/height ratios of page frame
and map
if [[ $( awk "BEGIN { print ( ( "${LAT_HEIGHT}" * "${MAX_WIDTH}" ) / ( "${LON_WIDTH}" *
"${MAX_HEIGHT}" ) ) > 1 ) }" ) == "1" ]] ; then
    # MapAnnotation derived from height
    mpa=$( awk "BEGIN { printf(\"%1.0e\n\", "${LAT_HEIGHT}" / "${MAP_TICK_MARKS}");
}")
    WIDTH=$( awk "BEGIN { print(("${LON_WIDTH}" *
"${MAX_HEIGHT}"/"${LAT_HEIGHT}"); }")

    WIDTH_TEST=$( awk "BEGIN { print( "$WIDTH" < "${MIN_COLORBAR_WIDTH}" ); }" )
    if [ "${WIDTH_TEST}" -ne "0" ] ; then
        COLORBAR_WIDTH=${MIN_COLORBAR_WIDTH}
    else
        COLORBAR_WIDTH=$WIDTH
    fi
else
    # MapAnnotation derived from width
    mpa=$( awk "BEGIN { printf(\"%1.0e\n\", "${LON_WIDTH}" /
"${MAP_TICK_MARKS}"); }")
    WIDTH=${MAX_WIDTH}
    COLORBAR_WIDTH=$WIDTH
fi

# calculate stepwidth of tick marks (mpa) and sub tickmarks (mpf) for the Map
Annotations
mpa_exponent=${mpa#*e}
mpa_mantisse=${mpa%e*}
if [ "${mpa_mantisse}" -ge "3" ] ; then
    mpf=$((mpa_mantisse / 3))
    mpa=$((mpf * 3))"e"${mpa_exponent}
    mpf=$mpf"e"${mpa_exponent}
else
    mpf="5e"$((${mpa_exponent} - 1))
fi

MapAnnotation="a"$mpa"f"$mpf"/a"$mpa"f"$mpf

# some further things for correct placement
HalfWIDTH=$( awk "BEGIN { print("${WIDTH}" / 2); }")

#####
### Preparation Phase
#####

```

```

# Prepare workspace by creating needed directories
mkdir -p $SCRATCHDIR
if [ "$TARGET_SCP_ADDRESS" == "" ] ; then
    # Produce output in final local dir. Create it, if it does not exist.
    if [ ! -d $TGTDIR ] ; then
        mkdir -p $TGTDIR
    fi
fi
# enter working directory
cd $SCRATCHDIR

# set gmt options (file ".gmtdefaults4")
gmtset BASEMAP_TYPE          plain
gmtset PAGE_ORIENTATION     $ORIENTATION
gmtset PAPER_MEDIA          $PAPER
gmtset ANNOT_FONT_SIZE     $ANNOTATION_FONT_SIZE
gmtset LABEL_FONT_SIZE     $LABEL_FONT_SIZE
gmtset UNIX_TIME_POS       $( awk "BEGIN { print( ("WIDTH" -
"$COLORBAR_WIDTH") / 2) }")c/-4.5c

# create color bar:
GridInfo=`grdinfo -C "${INFILE}?${Species}[0,0]"`
min_value=`echo $GridInfo | cut -f 6 -d" "`
max_value=`echo $GridInfo | cut -f 7 -d" "`
steps_value=$( awk "BEGIN { print(("max_value" - "min_value")/100); }" )

# echo "*** - palette file..." 1>&2
makecpt -Cseis -I -T$min_value/$max_value/$steps_value > $PALFILE

cba=a$( awk "BEGIN { print(("max_value" - "min_value")/5); }")
ColorBarAnnotation=${cba%.*}e${cba##*e}

#####
### Render Phase
#####
# echo "*** plot one field from netCDF file..." 1>&2
grd2xyz ${INFILE}?${Species}[0,0] >> $SCRATCHDIR/tmp.$$asc
pscontour $SCRATCHDIR/tmp.$$asc -P -R$PlotRange -I -J$PROJECTION/$WIDTH -C
$PALFILE -K -Xc -Y$Y_SHIFT -U$PLOT_NETCDF_VERSION > $TMPPS
rm -f $SCRATCHDIR/tmp.$$asc

# echo "*** overlay coast lines..." 1>&2
pscoast -J$PROJECTION/$WIDTH -B${MapAnnotation}nSEW -R -0 -K -P -Dh -W1 -A0/0/1
>> $TMPPS

# echo "*** place headline..." 1>&2
echo $HeadlinePosition' 16 0 4 CB '$HeadlineText | pstext -R -J$PROJECTION/$WIDTH
-0 -K -P -N -D0c/0.5c >> $TMPPS

# echo "*** place color bar..." 1>&2
psscale -C$PALFILE -D$HalfWIDTH/-1.5/${COLORBAR_WIDTH}c/0.5ch -0 -P -B
${ColorBarAnnotation}:"$Unit": >> $TMPPS

# echo "*** convert to PNG..." 1>&2
ps2raster $TMPPS -A -Tg -E$PNG_RESOLUTION

# echo "*** move generated PNG to final destination."
mv ${TMPPS%.*}.png $OUTFILE

## SCP feature disabled
#if [ "$TARGET_SCP_ADDRESS" != "" ] ; then
#    scp ${TMPPS%.*}.png $TARGET_SCP_ADDRESS:$OUTFILE

```

```

#      rm ${TMPPS%.*}.png
#else
#      mv ${TMPPS%.*}.png $OUTFILE
#fi

#####
### Metadata Phase
#####

# Only deal with metadata if an input meta file was specified and exists
if [ -n $METAFILE ] ; then
    if [ -e $METAFILE ] ; then
        # copy the metafile to output location
        outpath=${OUTFILE%/*}
        mfname=${METAFILE##*/}
        OUTMETA=$outpath/$mfname
        cp $METAFILE $OUTMETA
        #
        # TODO: perform metadata update here currently done externally
        # xmlstarlet ...
        #
    fi
fi

#####
### Cleaning Phase
#####

# echo "** cleaning up..." 1>&2
rm $PALFILE
rm $TMPPS
rm .gmtdefaults4
rm .gmtcommands4
cd $SCRATCHIDIR
cd ..
rmdir --ignore-fail-on-non-empty $SCRATCHDIR

# echo "** Plot finished" 1>&2
exit 0

```

7.3. Metadata for CWF Data

7.3.1. Metadata Template

```

<?xml version="1.0" encoding="utf-8"?>
<MD_Metadata xmlns="http://www.isotc211.org/2005/gmd"
  xmlns:gco="http://www.isotc211.org/2005/gco"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:gmd="http://www.isotc211.org/2005/gmd"
  xsi:schemaLocation="http://www.isotc211.org/2005/gmd
http://www.isotc211.org/2005/gmd/metadataEntity.xsd"
  id="de.dlr.ipa.ECHAM5MESSy.CWF">
  <fileIdentifier xmlns:p="http://www.orbeon.com/oxf/pipeline"
    xmlns:oxf="http://www.orbeon.com/oxf/processors">
    <gco:CharacterString
      xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">

```

```

        de.dlr.ipa.ECHAM5MESSy.CWF
    </gco:CharacterString>
</fileIdentifier>
<language xmlns:p="http://www.orbeon.com/oxf/pipeline"
  xmlns:oxf="http://www.orbeon.com/oxf/processors">
  <gco:CharacterString>en</gco:CharacterString>
</language>
<hierarchyLevel xmlns:p="http://www.orbeon.com/oxf/pipeline"
  xmlns:oxf="http://www.orbeon.com/oxf/processors">
  <MD_ScopeCode
    codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_ScopeCode"

```

```

codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_ScopeCode_m
odel">

```

```

    model
  </MD_ScopeCode>
</hierarchyLevel>
<hierarchyLevelName xmlns:p="http://www.orbeon.com/oxf/pipeline"
  xmlns:oxf="http://www.orbeon.com/oxf/processors">
  <gco:CharacterString
    xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
    ECHAM5MESSy.NUDGED.FORECAST
  </gco:CharacterString>
</hierarchyLevelName>
<contact xmlns:p="http://www.orbeon.com/oxf/pipeline"
  xmlns:oxf="http://www.orbeon.com/oxf/processors">
  <CI_ResponsibleParty>
    <individualName>
      <gco:CharacterString>
        Johannes Hendricks
      </gco:CharacterString>
    </individualName>
    <organisationName>
      <gco:CharacterString
        xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
        DLR, IPA
      </gco:CharacterString>
    </organisationName>
    <contactInfo>
      <CI_Contact>
        <address>
          <CI_Address>
            <electronicMailAddress>
              <gco:CharacterString
                xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
                Johannes.Hendricks@dlr.de
              </gco:CharacterString>
            </electronicMailAddress>
          </CI_Address>
        </address>
        <onlineResource>
          <CI_OnlineResource>
            <linkage>
              <URL
                xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
                http://www.dlr.de/pa/
              </URL>
            </linkage>

```

```
<function>
  <CI_OnLineFunctionCode
```

```
codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#CI_OnLineFunctionCode"
```

```
codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#CI_OnLineFunctionCode_download">
```

```
        information
      </CI_OnLineFunctionCode>
    </function>
  </CI_OnlineResource>
</onlineResource>
</CI_Contact>
</contactInfo>
<role>
  <CI_RoleCode
```

```
codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#CI_RoleCode"
```

```
codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#CI_RoleCode_resourceProvider">
```

```
        resourceProvider
      </CI_RoleCode>
    </role>
  </CI_ResponsibleParty>
</contact>
<dateStamp xmlns:p="http://www.orbeon.com/oxf/pipeline"
  xmlns:oxf="http://www.orbeon.com/oxf/processors">
  <gco:DateTime
    xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
    2007-05-25T16:00:00
  </gco:DateTime>
</dateStamp>
<metadataStandardName xmlns:p="http://www.orbeon.com/oxf/pipeline"
  xmlns:oxf="http://www.orbeon.com/oxf/processors">
  <gco:CharacterString>ISO 19115</gco:CharacterString>
</metadataStandardName>
<metadataStandardVersion
  xmlns:p="http://www.orbeon.com/oxf/pipeline"
  xmlns:oxf="http://www.orbeon.com/oxf/processors">
  <gco:CharacterString>
    ISO 19139 / C3Grid Profile V1.0
  </gco:CharacterString>
</metadataStandardVersion>
<dataSetURI xmlns:p="http://www.orbeon.com/oxf/pipeline"
  xmlns:oxf="http://www.orbeon.com/oxf/processors">
  <gco:CharacterString>
```

```
gsiftp://gridftp.dkrz.de/prj/bb0300/work/DLR_IPA/raw_data/forecast__200605.18_tracer_gp.nc
```

```
  </gco:CharacterString>
</dataSetURI>
<locale xmlns:p="http://www.orbeon.com/oxf/pipeline"
  xmlns:oxf="http://www.orbeon.com/oxf/processors">
  <PT_Locale id="de">
    <languageCode>
```

```

        <LanguageCode
codeList="http://wis.wmo.int/2006/catalogues/ML_gmxCodelists.xml#LanguageCode"
        codeListValue="de">
            de
        </LanguageCode>
    </languageCode>
    <country>
        <Country
codeList="http://wis.wmo.int/2006/catalogues/ML_gmxCodelists.xml#Country"
        codeListValue="Germany">
            Germany
        </Country>
    </country>
    <characterEncoding>
        <MD_CharacterSetCode
codeList="http://wis.wmo.int/2006/catalogues/ML_gmxCodelists.xml#MD_CharacterSetC
ode"
        codeListValue="8859part1">
            8859part1
        </MD_CharacterSetCode>
    </characterEncoding>
</PT_Locale>
</locale>
<referenceSystemInfo xmlns:p="http://www.orbeon.com/oxf/pipeline"
    xmlns:oxf="http://www.orbeon.com/oxf/processors">
    <MD_ReferenceSystem>
        <referenceSystemIdentifier>
            <RS_Identifier>
                <code>
                    <gco:CharacterString />
                </code>
                <codeSpace>
                    <gco:CharacterString>
                        C3Grid
                    </gco:CharacterString>
                </codeSpace>
            </RS_Identifier>
        </referenceSystemIdentifier>
    </MD_ReferenceSystem>
</referenceSystemInfo>
<identificationInfo xmlns:p="http://www.orbeon.com/oxf/pipeline"
    xmlns:oxf="http://www.orbeon.com/oxf/processors">
    <MD_DataIdentification>
        <citation>
            <CI_Citation>
                <title>
                    <gco:CharacterString
xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
                        Chemical Weather Forecast Workflow
                    </gco:CharacterString>
                </title>
                <alternateTitle>
                    <gco:CharacterString />
                </alternateTitle>
                <date>

```

```

    <CI_Date>
      <date>
        <gco:Date
xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
          2007-05-25
        </gco:Date>
      </date>
    </dateType>
    <CI_DateTypeCode

```

codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#CI_DateTypeCode"

codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#CI_DateTypeCode_creation"

```

      creation
    </CI_DateTypeCode>
  </dateType>
</CI_Date>
</date>
<identifier>
  <MD_Identifier>
    <code>
      <gco:CharacterString />
    </code>
  </MD_Identifier>
</identifier>
<citedResponsibleParty>
  <CI_ResponsibleParty
    id="Responsible_Citation">
    <individualName>
      <gco:CharacterString />
    </individualName>
    <organisationName>
      <gco:CharacterString

```

```

xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
      DLR-PA
    </gco:CharacterString>
  </organisationName>
  <contactInfo>
    <CI_Contact>
      <address>
        <CI_Address>

```

<electronicMailAddress>

<gco:CharacterString />

</electronicMailAddress>

```

      </CI_Address>
    </address>
  </CI_Contact>
</contactInfo>
<role>
  <CI_RoleCode

```

codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#CI_RoleCode"

```
codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#CI_RoleCode_re  
sourceProvider">
```

```
        resourceProvider  
        </CI_RoleCode>  
    </role>  
    </CI_ResponsibleParty>  
</citedResponsibleParty>  
</CI_Citation>  
</citation>  
<abstract>  
    <gco:CharacterString  
        xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">  
        Chemical Weather Forecasts from ECHAM5/MESSy  
    </gco:CharacterString>  
</abstract>  
<status>  
    <MD_ProgressCode
```

```
codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_ProgressCode"
```

```
codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_ProgressCod  
e_completed">
```

```
        completed  
    </MD_ProgressCode>  
</status>  
<pointOfContact>  
    <CI_ResponsibleParty id="Responsible_Contact">  
        <individualName>  
            <gco:CharacterString
```

```
xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">  
                Johannes Hendricks  
            </gco:CharacterString>  
        </individualName>  
        <organisationName>  
            <gco:CharacterString
```

```
xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">  
                DLR-PA  
            </gco:CharacterString>  
        </organisationName>  
        <contactInfo>  
            <CI_Contact>  
                <address>  
                    <CI_Address>  
                        <electronicMailAddress>  
                            <gco:CharacterString
```

```
xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
```

```
Johannes.Hendricks@dlr.de
```

```
</gco:CharacterString>
```

```
                            </electronicMailAddress>  
                        </CI_Address>  
                    </address>  
            </CI_Contact>
```

```

    </contactInfo>
    <role>
      <CI_RoleCode

```

codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#CI_RoleCode"

codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#CI_RoleCode_pointOfContact">

```

      pointOfContact
        </CI_RoleCode>
      </role>
    </CI_ResponsibleParty>
  </pointOfContact>
  <resourceMaintenance>
    <MD_MaintenanceInformation>
      <maintenanceAndUpdateFrequency>
        <MD_MaintenanceFrequencyCode

```

codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_MaintenanceFrequencyCode"

codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_MaintenanceFrequencyCode_asNeeded">

```

      asNeeded
        </MD_MaintenanceFrequencyCode>
      </maintenanceAndUpdateFrequency>
    <maintenanceNote>
      <gco:CharacterString />
    </maintenanceNote>
  </MD_MaintenanceInformation>
</resourceMaintenance>
<graphicOverview>
  <MD_BrowseGraphic>
    <fileName>
      <gco:CharacterString />
    </fileName>
    <fileDescription>
      <gco:CharacterString />
    </fileDescription>
    <fileType>
      <gco:CharacterString />
    </fileType>
  </MD_BrowseGraphic>
</graphicOverview>
<resourceFormat>
  <MD_Format>
    <name>
      <gco:CharacterString>
        netCDF
      </gco:CharacterString>
    </name>
    <version>
      <gco:CharacterString>3.5.0</gco:CharacterString>
    </version>
  </MD_Format>
</resourceFormat>
<descriptiveKeywords>

```

```

    <MD_Keywords>
      <keyword>
        <gco:CharacterString
xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
          chemical weather forecast
        </gco:CharacterString>
      </keyword>
    </type>
    <MD_KeywordTypeCode

```

codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_KeywordTypeCode"

codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_KeywordTypeCode_theme">

```

      theme
    </MD_KeywordTypeCode>
  </type>
</MD_Keywords>
</descriptiveKeywords>
<resourceConstraints>
  <MD_LegalConstraints>
    <accessConstraints>
      <MD_RestrictionCode

```

codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_RestrictionCode"

codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_RestrictionCode_copyright">

```

      copyright
    </MD_RestrictionCode>
  </accessConstraints>
  <useConstraints>
    <MD_RestrictionCode

```

codeListValue="http://www.c3grid.de/catalogues/gmxCodelists.xml#MD_RestrictionCode_c3member"

codeList="http://www.c3grid.de/catalogues/gmxCodelists.xml#MD_RestrictionCode">

```

      c3member
    </MD_RestrictionCode>
  </useConstraints>
  </MD_LegalConstraints>
</resourceConstraints>
<spatialRepresentationType>
  <MD_SpatialRepresentationTypeCode

```

codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_SpatialRepresentationTypeCode"

codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_SpatialRepresentationCode_vector">

```

        vector
        </MD_SpatialRepresentationTypeCode>
</spatialRepresentationType>
<language>
    <LanguageCode

```

```

codeListValue="http://wis.wmo.int/2006/catalogues/ML_gmxCodelists.xml#LanguageCode_eng"

```

```

codeList="http://wis.wmo.int/2006/catalogues/ML_gmxCodelists.xml#LanguageCode">
    eng
    </LanguageCode>
</language>
<topicCategory>

```

```

<MD_TopicCategoryCode>climatologyMeteorologyAtmosphere</MD_TopicCategoryCode>
</topicCategory>
<environmentDescription>
    <gco:CharacterString>
        C3Grid: Chemical weather forecast, implementation
        DLR
    </gco:CharacterString>
</environmentDescription>
<extent>
    <EX_Extent>
        <geographicElement>
            <EX_GeographicBoundingBox>
                <westBoundLongitude>
                    <gco:Decimal

```

```

xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
    0
    </gco:Decimal>
</westBoundLongitude>
<eastBoundLongitude>
    <gco:Decimal

```

```

xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
    360
    </gco:Decimal>
</eastBoundLongitude>
<southBoundLatitude>
    <gco:Decimal

```

```

xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
    -90
    </gco:Decimal>
</southBoundLatitude>
<northBoundLatitude>
    <gco:Decimal

```

```

xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
    90
    </gco:Decimal>
</northBoundLatitude>
</EX_GeographicBoundingBox>
</geographicElement>
<temporalElement>

```

```

        <EX_TemporalExtent>
            <extent>
                <gml:TimePeriod
                    gml:id="templatetimeperiod">
                        <gml:beginPosition
                            xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
                                2006-05-18T00:00:00Z
                            </gml:beginPosition>
                            <gml:endPosition
                                xmlns:xx-
                                    forms="http://orbeon.org/oxf/xml/xforms">
                                        2006-05-18T21:00:00Z
                                    </gml:endPosition>
                                <gml:timeInterval
                                    xmlns:xxforms="http://orbeon.org/oxf/xml/xforms" factor="0"
                                    unit="hour"
                                    radix="1">
                                        3
                                    </gml:timeInterval>
                                </gml:TimePeriod>
                            </extent>
                        </EX_TemporalExtent>
                    </temporalElement>
                <verticalElement>
                    <EX_VerticalExtent>
                        <minimumValue>
                            <gco:Real
                                xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
                                    99999
                                </gco:Real>
                            </minimumValue>
                            <maximumValue>
                                <gco:Real
                                    xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
                                        10
                                    </gco:Real>
                                </maximumValue>
                            <verticalCRS
                                xlink:href="http://www.c3grid.de/files/schemas/C3grid-crs-
                                dict.xml#vertCRS.standardPressureLevel"
                                xlink:title="CRS for Height"
                                xlink:role="add_info" />
                            </EX_VerticalExtent>
                        </verticalElement>
                    </EX_Extent>
                </extent>
            </MD_DataIdentification>
        </identificationInfo>
        <contentInfo xmlns:p="http://www.orbeon.com/oxf/pipeline"
            xmlns:oxf="http://www.orbeon.com/oxf/processors">
            <MD_CoverageDescription id="any_id">
                <attributeDescription
                    <gco:RecordType
                        xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
                            mole_fraction_of_ozone_in_air
                    </gco:RecordType>
                </attributeDescription>
            </MD_CoverageDescription>
        </contentInfo>
    </gml:CoverageDescription>
</gml:CoverageDescription>

```

```

        </gco:RecordType>
    </attributeDescription>
    <contentType>
        <MD_CoverageContentTypeCode

```

codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_CoverageContentT
ypeCode"

codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_CoverageCon
tentTypeCode_thematicClassification">

```

        thematicClassification
    </MD_CoverageContentTypeCode>
</contentType>
<dimension>
    <MD_RangeDimension>
        <descriptor>
            <gco:CharacterString

```

xmlns:xforms="http://orbeon.org/oxf/xml/xforms">

```

                1
            </gco:CharacterString>
        </descriptor>
    </MD_RangeDimension>
</dimension>
</MD_CoverageDescription>
</contentInfo>
<gmd:contentInfo xmlns:xforms="http://orbeon.org/oxf/xml/xforms"
    xmlns:p="http://www.orbeon.com/oxf/pipeline"
    xmlns:oxf="http://www.orbeon.com/oxf/processors">
    <gmd:MD_CoverageDescription>
        <gmd:attributeDescription>
            <gco:RecordType>
                mole_fraction_of_nitrogen_oxide_in_air
            </gco:RecordType>
        </gmd:attributeDescription>
        <gmd:contentType>
            <gmd:MD_CoverageContentTypeCode

```

codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_CoverageCon
tentTypeCode_thematicClassification"

codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_CoverageContentT
ypeCode">

```

        thematicClassification
    </gmd:MD_CoverageContentTypeCode>
</gmd:contentType>
<gmd:dimension>
    <gmd:MD_RangeDimension>
        <gmd:descriptor>
            <gco:CharacterString>1</gco:CharacterString>
        </gmd:descriptor>
    </gmd:MD_RangeDimension>
</gmd:dimension>
</gmd:MD_CoverageDescription>
</gmd:contentInfo>
<gmd:contentInfo xmlns:xforms="http://orbeon.org/oxf/xml/xforms"
    xmlns:p="http://www.orbeon.com/oxf/pipeline"

```

```

xmlns:oxf="http://www.orbeon.com/oxf/processors">
<gmd:MD_CoverageDescription>
  <gmd:attributeDescription>
    <gco:RecordType>
      mole_fraction_of_nitrogen_dioxide_in_air
    </gco:RecordType>
  </gmd:attributeDescription>
  <gmd:contentType>
    <gmd:MD_CoverageContentTypeCode

```

codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_CoverageContentTypeCode_thematicClassification"

codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_CoverageContentTypeCode">

```

      thematicClassification
    </gmd:MD_CoverageContentTypeCode>
  </gmd:contentType>
  <gmd:dimension>
    <gmd:MD_RangeDimension>
      <gmd:descriptor>
        <gco:CharacterString>1</gco:CharacterString>
      </gmd:descriptor>
    </gmd:MD_RangeDimension>
  </gmd:dimension>
</gmd:MD_CoverageDescription>
</gmd:contentInfo>
<gmd:contentInfo xmlns:xxforms="http://orbeon.org/oxf/xml/xforms"
  xmlns:p="http://www.orbeon.com/oxf/pipeline"
  xmlns:oxf="http://www.orbeon.com/oxf/processors">
  <gmd:MD_CoverageDescription>
    <gmd:attributeDescription>
      <gco:RecordType>
        mole_fraction_of_carbon_monoxide_in_air
      </gco:RecordType>
    </gmd:attributeDescription>
    <gmd:contentType>
      <gmd:MD_CoverageContentTypeCode

```

codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_CoverageContentTypeCode_thematicClassification"

codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_CoverageContentTypeCode">

```

      thematicClassification
    </gmd:MD_CoverageContentTypeCode>
  </gmd:contentType>
  <gmd:dimension>
    <gmd:MD_RangeDimension>
      <gmd:descriptor>
        <gco:CharacterString>1</gco:CharacterString>
      </gmd:descriptor>
    </gmd:MD_RangeDimension>
  </gmd:dimension>
</gmd:MD_CoverageDescription>
</gmd:contentInfo>
<gmd:contentInfo xmlns:xxforms="http://orbeon.org/oxf/xml/xforms"

```

```

xmlns:p="http://www.orbeon.com/oxf/pipeline"
xmlns:oxf="http://www.orbeon.com/oxf/processors">
<gmd:MD_CoverageDescription>
  <gmd:attributeDescription>
    <gco:RecordType>
      mole_fraction_of_peroxy_acetyl_nitrate_in_air
    </gco:RecordType>
  </gmd:attributeDescription>
  <gmd:contentType>
    <gmd:MD_CoverageContentTypeCode

```

codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_CoverageContentTypeCode_thematicClassification"

codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_CoverageContentTypeCode">

```

      thematicClassification
    </gmd:MD_CoverageContentTypeCode>
  </gmd:contentType>
  <gmd:dimension>
    <gmd:MD_RangeDimension>
      <gmd:descriptor>
        <gco:CharacterString>1</gco:CharacterString>
      </gmd:descriptor>
    </gmd:MD_RangeDimension>
  </gmd:dimension>
</gmd:MD_CoverageDescription>
</gmd:contentInfo>
<gmd:contentInfo xmlns:xxforms="http://orbeon.org/oxf/xml/xforms"
  xmlns:p="http://www.orbeon.com/oxf/pipeline"
  xmlns:oxf="http://www.orbeon.com/oxf/processors">
  <gmd:MD_CoverageDescription>
    <gmd:attributeDescription>
      <gco:RecordType>
        mole_fraction_of_hydroxyl_radical_in_air
      </gco:RecordType>
    </gmd:attributeDescription>
    <gmd:contentType>
      <gmd:MD_CoverageContentTypeCode

```

codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_CoverageContentTypeCode_thematicClassification"

codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_CoverageContentTypeCode">

```

      thematicClassification
    </gmd:MD_CoverageContentTypeCode>
  </gmd:contentType>
  <gmd:dimension>
    <gmd:MD_RangeDimension>
      <gmd:descriptor>
        <gco:CharacterString>1</gco:CharacterString>
      </gmd:descriptor>
    </gmd:MD_RangeDimension>
  </gmd:dimension>
</gmd:MD_CoverageDescription>
</gmd:contentInfo>

```

```

<gmd:contentInfo xmlns:xxforms="http://orbeon.org/oxf/xml/xforms"
  xmlns:p="http://www.orbeon.com/oxf/pipeline"
  xmlns:oxf="http://www.orbeon.com/oxf/processors">
  <gmd:MD_CoverageDescription>
    <gmd:attributeDescription>
      <gco:RecordType>
        mole_fraction_of_hydroperoxy_radical_in_air
      </gco:RecordType>
    </gmd:attributeDescription>
    <gmd:contentType>
      <gmd:MD_CoverageContentTypeCode

```

```

codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_CoverageContentTypeCode_thematicClassification"

```

```

codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_CoverageContentTypeCode">

```

```

      thematicClassification
    </gmd:MD_CoverageContentTypeCode>
  </gmd:contentType>
  <gmd:dimension>
    <gmd:MD_RangeDimension>
      <gmd:descriptor>
        <gco:CharacterString>1</gco:CharacterString>
      </gmd:descriptor>
    </gmd:MD_RangeDimension>
  </gmd:dimension>
</gmd:MD_CoverageDescription>
</gmd:contentInfo>
<gmd:contentInfo xmlns:xxforms="http://orbeon.org/oxf/xml/xforms"
  xmlns:p="http://www.orbeon.com/oxf/pipeline"
  xmlns:oxf="http://www.orbeon.com/oxf/processors">
  <gmd:MD_CoverageDescription>
    <gmd:attributeDescription>
      <gco:RecordType>surface_air_pressure</gco:RecordType>
    </gmd:attributeDescription>
    <gmd:contentType>
      <gmd:MD_CoverageContentTypeCode

```

```

codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_CoverageContentTypeCode_thematicClassification"

```

```

codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_CoverageContentTypeCode">

```

```

      thematicClassification
    </gmd:MD_CoverageContentTypeCode>
  </gmd:contentType>
  <gmd:dimension>
    <gmd:MD_RangeDimension>
      <gmd:descriptor>
        <gco:CharacterString>Pa</gco:CharacterString>
      </gmd:descriptor>
    </gmd:MD_RangeDimension>
  </gmd:dimension>
</gmd:MD_CoverageDescription>
</gmd:contentInfo>
<gmd:contentInfo xmlns:xxforms="http://orbeon.org/oxf/xml/xforms"

```

```

xmlns:p="http://www.orbeon.com/oxf/pipeline"
xmlns:oxf="http://www.orbeon.com/oxf/processors">
<gmd:MD_CoverageDescription>
  <gmd:attributeDescription>
    <gco:RecordType>geopotential</gco:RecordType>
  </gmd:attributeDescription>
  <gmd:contentType>
    <gmd:MD_CoverageContentTypeCode

```

```

codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_CoverageCon
tentTypeCode_thematicClassification"

```

```

codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_CoverageContentT
ypeCode">

```

```

      thematicClassification
    </gmd:MD_CoverageContentTypeCode>
  </gmd:contentType>
  <gmd:dimension>
    <gmd:MD_RangeDimension>
      <gmd:descriptor>
        <gco:CharacterString>
          m2 s-2
        </gco:CharacterString>
      </gmd:descriptor>
    </gmd:MD_RangeDimension>
  </gmd:dimension>
</gmd:MD_CoverageDescription>
</gmd:contentInfo>
<distributionInfo xmlns:p="http://www.orbeon.com/oxf/pipeline"
  xmlns:oxf="http://www.orbeon.com/oxf/processors">
  <MD_Distribution>
    <distributionFormat>
      <MD_Format>
        <name>
          <gco:CharacterString
            netCDF
          </gco:CharacterString>
        </name>
        <version>
          <gco:CharacterString
            3.5.0
          </gco:CharacterString>
        </version>
      </MD_Format>
    </distributionFormat>
    <distributor>
      <MD_Distributor>
        <distributorContact>
          <CI_ResponsibleParty>
            <organisationName>
              <gco:CharacterString
                DLR-IPA
              </gco:CharacterString>
            </gco:CharacterString>
          </CI_ResponsibleParty>
        </distributorContact>
      </MD_Distributor>
    </distributor>
  </MD_Distribution>
</distributionInfo>

```

```

        </organisationName>
        <role>
            <CI_RoleCode
codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#CI_RoleCode"
codeListValue="resourceProvider">
                resourceProvider
                </CI_RoleCode>
            </role>
        </CI_ResponsibleParty>
    </distributorContact>
    <distributorTransferOptions>
        <MD_DigitalTransferOptions>
            <transferSize>
                <gco:Real
xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
                    232
                </gco:Real>
            </transferSize>
            <onLine>
                <CI_OnlineResource>
                    <linkage>
                        <URL
codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#CI_OnlineFunctionCo
de"
codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#CI_OnlineFunct
ionCode_download">
                            </URL>
                        </linkage>
                        <protocol>
                            <gco:CharacterString
/>
                                </protocol>
                                <applicationProfile>
                                    <gco:CharacterString
/>
                                        </applicationProfile>
                                        <name>
                                            <gco:CharacterString
/>
                                                </name>
                                                <description>
                                                    <gco:CharacterString
/>
                                                        </description>
                                                        <function>
<CI_OnlineFunctionCode
                                </function>
                                </CI_OnlineFunctionCode>
codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#CI_OnlineFunctionCo
de"
codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#CI_OnlineFunct
ionCode_download">
                                    download
                                </CI_OnlineFunctionCode>

```



```

</distributionInfo>
<dataQualityInfo xmlns:p="http://www.orbeon.com/oxf/pipeline"
  xmlns:oxf="http://www.orbeon.com/oxf/processors">
  <DQ_DataQuality>
    <scope>
      <DQ_Scope>
        <level>
          <MD_ScopeCode

```

```
codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_ScopeCode"
```

```
codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_ScopeCode_dataset">
```

```

          dataset
        </MD_ScopeCode>
      </level>
    </DQ_Scope>
  </scope>
  <lineage>
    <LI_Lineage>
      <statement>
        <gco:CharacterString />
      </statement>
      <processStep>
        <LI_ProcessStep id="Step_0">
          <description>
            <gco:CharacterString

```

```
xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
```

```
C3Grid: Chemical weather
```

```
forecast,
```

```

          implementation DLR
        </gco:CharacterString>
      </description>
      <rationale>
        <gco:CharacterString />
      </rationale>
      <dateTime>
        <gco:DateTime

```

```
xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
```

```
2007-05-25T00:00:00
```

```

        </gco:DateTime>
      </dateTime>
      <processor>
        <CI_ResponsibleParty>
          <role>
            <CI_RoleCode

```

```
codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#CI_RoleCode_author"
```

```
codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#CI_RoleCode">
```

```

          author
        </CI_RoleCode>
      </role>
    </CI_ResponsibleParty>
  </processor>
</source>

```

```

        <LI_Source>
            <description>
                <gco:CharacterString
/>
            </description>
        </LI_Source>
    </source>
</LI_ProcessStep>
</processStep>
</LI_Lineage>
</lineage>
</DQ_DataQuality>
</dataQualityInfo>
<metadataMaintenance xmlns:p="http://www.orbeon.com/oxf/pipeline"
    xmlns:oxf="http://www.orbeon.com/oxf/processors">
    <MD_MaintenanceInformation>
        <maintenanceAndUpdateFrequency>
            <MD_MaintenanceFrequencyCode

```

```
codeList="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_MaintenanceFrequencyCode"
```

```
codeListValue="http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_ScopeCode_asNeeded">
```

```

            asNeeded
        </MD_MaintenanceFrequencyCode>
    </maintenanceAndUpdateFrequency>
    <maintenanceNote>
        <gco:CharacterString />
    </maintenanceNote>
</MD_MaintenanceInformation>
</metadataMaintenance>
</MD_Metadata>

```

7.3.2. Metadata Update XSL for CWF Result Data

```

<xsl:stylesheet version="1.0" xmlns="http://www.isotc211.org/2005/gmd"
xmlns:ind="http://c3grid.de/webservice/common/"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:gco="http://www.isotc211.org/2005/gco"
xmlns:gml="http://www.opengis.net/gml"
xmlns:iso="http://www.isotc211.org/2005/gmd" xmlns:date="http://exslt.org/dates-
and-times" exclude-result-prefixes="xsl gml gco iso date ind">

```

```
<!--
```

```

META_WRITEISO.XSL: ISO metadata update for ECHAM5/MESSy Chemical Weather Forecast
CALL: xmlstarlet tr md_update_cwf.xsl -s size=237322188 -s start=2006-06-
18T00:00:00Z -s stop=2006-06-18T23:59:59Z -s version='C3Grid ECHAM5/MESSy
Chemical Weather Forecast, version 0.1, 2008-07-01 (Johannes.Hendricks@dlr.de)'
-s constraints='$DATE+$FORECAST_LENGTH' de.dlr.ipa.ECHAM5MESSy.CWF.dlr.xml >
de.dlr.ipa.ECHAM5MESSy.CWF.dlr.$DATE_ $FORECAST_LENGTH.xml
VERSION: 0.1 based on META_WRITEISO.XSL by M.Stockhause, MPI-M
(martina.stockhause AT zmaw.de) - Version 0.1
HISTORY: creation on 2008-07-01
TODO: check consistency of identifiers
-->

```

```

<xsl:output method="xml" encoding="UTF-8" indent="yes"/>

<!-- remove empty lines from output file -->
<xsl:strip-space elements="iso:MD_Metadata"/>
<xsl:strip-space elements="iso:MD_DigitalTransferOptions"/>
<xsl:strip-space elements="iso:LI_Lineage"/>

<!-- old ISO IDs -->
<xsl:variable name="oldID">
  <xsl:value-of select="//iso:fileIdentifier/gco:CharacterString"/>
</xsl:variable>

<!-- new ISO ID -->
<!-- strip away everything after first dash, append dash and current time in
seconds -->
<xsl:variable name="newID">
  <xsl:choose>
    <xsl:when test="contains($oldID, '-')">
      <xsl:value-of select="concat(substring-before($oldID, '-'),
'- ',date:seconds())"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="concat($oldID, '-',date:seconds())"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:variable>

<xsl:variable name="postAbstract">
  <xsl:value-of select="' - Chemical Weather Forecast Model (ECHAM5/MESSy)'" />
</xsl:variable>

<xsl:variable name="actDateTime" select="substring-before(date:add(date:date-
time(),'-PT0H'),'Z')"/>
<xsl:variable name="actDate" select="substring-before(date:add(date:date-
time(),'-PT0H'),'T')"/>
<xsl:variable name="contentCode"
select="'http://wis.wmo.int/2006/catalogues/gmxCodelists.xml#MD_CoverageContentTy
peCode'"/>
<!-- MS, 2007-06-06: changed <xsl:variable name="contentCodeValue"
select="'thematicClassification'"/> -->
<xsl:variable name="contentCodeValue" select="'model'"/>

<xsl:variable name="newLinState">
  <xsl:value-of select="'ECHAM5/MESSy Chemical Weather Forecast'"/>
</xsl:variable>
<xsl:variable name="newLin" select="concat($version,' with Dataset: ')/>
<xsl:variable name="newLinOpt" select="' with ECHAM5/MESSy calls: '"/>

<!-- templates -->
<xsl:template match="@*|node()">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>

<!-- set new id in MD_Metadata tag -->
<xsl:template match="iso:fileIdentifier">

```

```

    <xsl:element name="fileIdentifier">
      <xsl:element name="gco:CharacterString">
        <xsl:value-of select="$newID"/>
      </xsl:element>
    </xsl:element>
  </xsl:template>
<xsl:template match="iso:MD_Metadata/@id">
  <xsl:attribute name="id"><xsl:value-of select="$newID"/></xsl:attribute>
</xsl:template>

<!-- set new dateStamp -->
<xsl:template match="iso:dateStamp/gco:DateTime/">
  <xsl:value-of select="$factDateTime"/>
</xsl:template>
<xsl:template match="iso:identificationInfo//iso:citation//iso:date//gco:Date/">
  <xsl:value-of select="$factDate"/>
</xsl:template>

<!-- abstract extended -->
<xsl:template match="iso:abstract/gco:CharacterString/">
  <xsl:value-of select="."/>
  <xsl:if test="not(contains(current(),$postAbstract))">
    <xsl:value-of select="$postAbstract"/>
  </xsl:if>
</xsl:template>

<!-- set file size of dataset -->
<xsl:template match="iso:transferSize//gco:Real/">
  <xsl:value-of select="$size"/>
</xsl:template>

<!-- set start and stop date of dataset -->
<xsl:template match="gml:beginPosition">
  <xsl:value-of select="$start" />
</xsl:template>
<xsl:template match="gml:endPosition">
  <xsl:value-of select="$stop" />
</xsl:template>

<!-- insert lineage info about processing done -->
<xsl:template match="iso:lineage//iso:statement/gco:CharacterString/">
  <xsl:value-of select="concat(.,', ',,$newLinState)"/>
</xsl:template>
<xsl:template match="iso:processStep">
  <xsl:variable name="no_proc" select="count(../iso:processStep)"/>
  <xsl:if test="position()=last()">
    <xsl:copy-of select="../iso:processStep"/>
    <xsl:element name="processStep">
      <xsl:element name="LI_ProcessStep">
        <xsl:attribute name="id">
          <xsl:value-of select="concat('Step_', $no_proc)"/>
        </xsl:attribute>
        <xsl:element name="description">
          <xsl:element name="gco:CharacterString">
            <xsl:value-of select="$newLin"/>
            '<xsl:value-of select="$oldids"/>' <xsl:value-of
select="$newLinOpt"/>
          <xsl:value-of select="$constraints"/>
        </xsl:element>
      </xsl:element>
    </xsl:element>
  </xsl:if>
</xsl:template>

```

```
<xsl:element name="dateTime">
  <xsl:element name="gco:DateTime">
    <xsl:value-of select="$actDateTime"/>
  </xsl:element>
</xsl:element>
<xsl:element name="source">
  <xsl:element name="LI_Source">
    <xsl:element name="description">
      <xsl:element name="gco:CharacterString">
        <xsl:value-of select="$oldids"/>
      </xsl:element>
    </xsl:element>
  </xsl:element>
</xsl:element>
</xsl:element>
</xsl:if>
</xsl:template>
</xsl:stylesheet>
```